Diseño de microcontrolador compatible con 80251

Proyecto fin de carrera
Ingeniería industrial

Universidad de Zaragoza
Centro Politécnico Superior
Departamento ingeniería electrónica y comunicaciones
Área de Tecnología electrónica

Autor
D. Fernando Gracia Royo

Director
Prof. José ignacio García Nicolás

Diseño de microcontrolador compatible con 80251

Resumen

En el presente proyecto hemos estudiado la arquitectura de la familia de microcontroladores 80251 de Intel con el fin de poder diseñar un IP en VHDL sintetizable del mismo. Basados en la arquitectura clásica de 8 Bits de su predecesora, la familia de microcontroladores 8051, incrementa su capacidad de computación con la extensión del tamaño de los datos con los que opera, pudiendo ser estos de 8, 16 y 32 Bits. Dichos microcontroladores trabajan con un conjunto de instrucciones tipo Cisc con instrucciones de distinto número de Bytes. Así mismo, dependiendo de la instrucción se invierten diferentes intervalos de tiempo en desarrollarlas. Cada instrucción está asociada a un número de ciclos de máquina dependiendo de la complejidad de la operación a realizar o los accesos a memoria que se deban realizar. La compatibilidad con programas diseñados para la arquitectura del 51 implica que el microprocesador pueda trabajar en dos modos: fuente y binario. Cada uno tiene una serie de instrucciones específicas y para poder trabajar con ellas en el otro modo se necesita un Opcode de escape, incrementando el tamaño de las instrucciones en un Byte. El estilo que siguen estos microprocesadores a la hora de apilar los datos en las memorias es del tipo big-endian, es decir, la parte de mayor peso de una palabra se coloca en la posición de la memoria más baja.

Nuestro objetivo ha sido el diseño del núcleo del microcontrolador 80251 en VHDL sintetizable para evaluar el número de puertas equivalentes. También se han realizado mejoras de diseño para minimizar el número de ciclos de máquina requeridos por las instrucciones. Para lograrlo se ha optimizado el acceso a las memorias para datos desalineados implementando una configuración de memorias que permite acceder a las memorias en un ciclo de reloj. En el desarrollo del diseño se ha tenido en cuenta el consumo de recursos de la FPGA realizando procesos de síntesis periódicos para asegurar un tamaño de hardware adecuado. La utilidad de este proyecto es reemplazar al 8051 de tres ciclos de reloj por ciclo de máquina. El motivo es el incremento velocidad de ejecución de programas al poder operar con datos de 1, 2 o 4 Bytes y aprovechar la reducción en el tamaño de código que permite obtener el conjunto de instrucciones adicionales.

Índice

Índice de figuras

Índice de tablas

Bibliografía

Intel corporation. “8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual”. 1996 ISBM: 272795-002.

Srefan Sjoholm, Lennart Lindh. “VHDL for Designer”. Prentice Hall. 1997 ISBM: 0-13-473414-9.

Roger Lipsett, Carl Schaefer, Cary Ussery. “VHDL: Hardware Desciption and Design”. KAP. 1989 ISBM: 0-7923-9030-X.

Capítulo 1: Introducción.

Este proyecto se ha realizado en el departamento de ingeniería Electrónica y Comunicaciones, área de Tecnología Electrónica. El objetivo es crear un IP sintetizable que pueda ser usado en futuros proyectos dentro de FPGA´s.

El objetivo de este proyecto es el diseño de un núcleo compatible con los microcontroladores de la familia 80251. Se busca tener un sustituto del microcontrolador 8051 de 8 bits con tres ciclos de reloj por ciclo de máquina que ya existe. El 251 es una arquitectura de 8 bits que es capaz de operar con datos de 8, 16 y 32 bits. La ampliación de tamaño de los datos permite ejecutar instrucciones más rápido y reducir el número de instrucciones necesarias para realizar una operación, reduciendo el código de los programas.

En este proyecto solo se diseñará el núcleo del microprocesador junto con las memorias para datos (RAM) y memorias para el código (ROM). No se diseñará ningún periférico ni habrá sistema de interrupciones. El diseño se realizará en VHDL con el fin de crear un IP sintetizable.

El diseño del microprocesador parte del estudio del funcionamiento del núcleo del microcontrolador a través de su manual técnico. De dicho estudio se ha deducido qué se deberá implementar en el diseño y qué partes quedan fuera del alcance por estar destinado al control de periféricos. Se ha tenido cuidado de no extendernos demasiado en el uso de recursos hardware, puesto que debería ser capaz de funcionar en una FPGA. Como mejora de diseño nos hemos planteado reducir el número de ciclos de máquina de las instrucciones, que dependen de cada instrucción y pueden ser de varios ciclos. El objetivo es conseguir que el máximo posible de instrucciones requieran un solo ciclo de

máquina y que este ciclo al mismo tiempo tenga solo dos ciclos de reloj.

El diseño será verificado mediante la ejecución de un programa escrito en código máquina. El proceso de ejecución se hará junto con un simulador de los microcontroladores 80251 y se hará el seguimiento de la ejecución de las instrucciones. Se utilizarán unos programas que realicen alguna operación matemática, sin necesidad de usar ningún periférico ni puerto. El resultado se comparará con el que resulta de ejecutar el mismo código pero compilado y ejecutado en el microcontrolador 8051, tanto en número de ciclos de reloj como en tamaño del programa en hexadecimal. Por otro lado se comparará el número de puertas equivalentes de ambos modelos para observar cuanto ha aumentado el hardware.

El diseño no parte desde cero, sino que al comienzo del mismo se me entregaron unos esquemáticos. Estos están jerarquizados en tres niveles y parten de una representación de la placa donde irían los componentes como el microcontrolador, memorias RAM externas o el clock. El segundo nivel contiene el núcleo del microcontrolador, las memorias internas y periféricos. El tercer nivel se corresponde al núcleo y es con el que hemos trabajado. En él hemos introducido una serie de bloques con las diferentes partes del núcleo. Los bloques son archivos en VHDL generados a partir de un esquemático con una determinada función. Comparten entre sí una serie de señales que permiten transmitirse datos.

Junto con los esquemáticos había también varios Packages. Los Packages son archivos en VHDL que contienen constantes, declaraciones de “Tipos” y funciones. Lo que se declara en un Package puede ser usado en cualquier bloque. En los Packages que se me han entregado contenían algunas constantes como los niveles de activación de Reset y Clock, constantes con Resets y direcciones de algunos registros. Por motivos de diseño algunos de estos Packages han sido modificados.

El contenido de esta memoria está dividido en tres partes y cinco capítulos. La primera parte es esta introducción. La segunda parte contiene tres capítulos y desarrolla las tres fases en las que se ha dividido la realización del proyecto. La última parte y capítulo son las conclusiones.

A modo de introducción, en las siguientes secciones se ha realizado un resumen de lo que nos encontraremos en cada uno de los capítulos de la segunda parte. Estos tres capítulos abordan la fase de estudio del funcionamiento de los microcontroladores 80251, la fase de diseño y por último la fase de pruebas.

1.1- Estudio del microcontrolador 80251.

En el capítulo 2 presentaremos los resultados del estudio del manual de los microcontroladores 80251. El estudio marcará la estructura que se deberá seguir en el diseño. Gracias a este estudio se ha dimensionado el tamaño de buses de datos, las memorias que se van a necesitar, la estructura de los bloques....

El estudio comienza con el origen de los microcontroladores 80251, que son una mejora de sus predecesores los 8051. Este dato es importante ya que condiciona toda la

arquitectura del 251 por motivos de compatibilidad. Cualquier programa y configuración de recursos que se haya realizado para la arquitectura del 51 debe funcionar en esta nueva arquitectura. Aunque ambos microcontroladores son de 8 bits, el 251 puede operar con datos de 16 y 32 bits a diferencia del 51 que solo puede operar con 8 bits.

Se repasa el espacio de memoria de la nueva arquitectura. Esta pasa de 128KBytes en el 51 hasta los 16 MByte que es capaz de acceder el 251. Para ello se ha ampliado el bus de direcciones a 24 bits. Para acceder a estas direcciones se incrementan los tipos de direccionamiento. La memoria interna también se amplía pasando de 256 Bytes a 512 o 1024 Bytes dependiendo del modelo.

Por otro lado se amplía el banco de registros y ahora se puede acceder a ellos como Bytes, Word y Double-words (8 bits, 16bits y 32 bits). Parte del banco de registros se comparte con la memoria RAM interna, pasando a ser la parte inferior de esta. Tanto en RAM como en los registros los datos siguen una estructura de almacenamiento en Big-endian.

Por otro lado se habla de los modos de operación, binario y fuente. Estos modos son necesarios para ser compatibles con programas del 51. La diferencia entre ambos modos es que el espacio de instrucciones cambia y para usar determinadas instrucciones se debe introducir un Byte de escape. Para trabajar con instrucciones nativas del 51 se usa el modo binario en donde los datos usados son de 8 bits. Las instrucciones introducidas en la arquitectura del 251 pueden trabajar con hasta 32 bits y el modo de trabajo es el fuente. El uso del Byte de escape permite trabajar con las instrucciones del otro modo en ambos modos de trabajo.

Las instrucciones siguen una tipología tipo Cisc. Los datos no están alineados con consecuencias negativas en la lectura del código al tener cada Opcode diferente número de Bytes.

1.2- Diseño del núcleo en VHDL.

El tercer capítulo trata sobre el proceso de diseño en VHDL del núcleo. Se comenta como se ha desarrollado el diseño. Comienza a partir del estudio del manual del microcontrolador y adopta sus conclusiones.

Además de la información suministrada del estudio del microcontrolador debemos conocer la estructura de funcionamiento. El funcionamiento interno consiste en leer los datos de código, decodificarlos, leer los datos de memoria, ejecutar la operación decodificada y escribir el resultado. No siempre es necesario pasar por las cinco fases. Sin embargo la forma de hacer estas fases es muy importante, ya que condicionan la duración del ciclo de máquina. Las fases o procesos se denominan Fetch, Decode, Read, Execute y Write. En esta sección se detalla el modo de elección de como se realiza cada proceso siendo el definitivo realizar un Fetch independiente y secuencialmente se realizan el Decode con el Read y Execute con el Write. Siguiendo esta estrategia conseguimos un ciclo de máquina con tan solo dos ciclos de reloj y un tiempo de ciclo razonable.

La configuración de las memorias adquiere en nuestro diseño especial importancia. Si queremos que las instrucciones se hagan en un ciclo de máquina el acceso a los datos se debe hacer en un ciclo de reloj. Pero los datos pueden ser de hasta 32bits y no están alineados. Se ha adoptado la configuración de dos memorias en paralelo con palabras de 32 bits (Double-Word) cada una que nos permiten leerlas y escribirlas en tan solo un ciclo de reloj.

Presentaremos una descripción de los bloques que hemos diseñado. Por lo general, los bloques se encargan de realizar uno de los procesos que antes hemos descrito. Sin embargo se han sacado dos funciones de estos bloques y se han colocado de forma independiente. Estos bloques se encargan de unos registros especiales llamados Program Status Word (PSW y PSW1) y de la operación división. La lectura y escritura se hacen desde el mimo bloque porque solo se pueden generar señales de control desde un bloque.

En total el diseño se ha realizado en seis bloques más Packages y otras entidades. El primer bloque se encarga del Fetch y apila las instrucciones que lee en una pila interna de 11 Bytes. También es el responsable de leer la memoria del código con algunas instrucciones. De aquí se envía al segundo bloque los Opcodes de las instrucciones que los decodifica. Aquí también se produce el control del resto de componentes. La principal función de este bloque es direccionar los datos en las memorias y mandarlos al proceso Execute para que opere con ellos. También se encarga de controlar la escritura del resultado. El tercer bloque que se describe es el de lectura y escritura de datos en la memoria interna o en los registros. El cuarto bloque es el encargado de realizar el Execute y se ha denominado Alu (Arithmetic Logic Unit). Por lo general, todas las operaciones son ejecutadas en este bloque salvo unas pocas que no hace falta, como las de salto que simplemente hay que informar al Fetch de la nueva dirección.

Las otras entidades contienen declaraciones de señales y sirven para conectar los bloques entre sí. Todas las señales que salen de un bloque deben estar reflejadas en este archivo, así como las señales que nos llegan del nivel anterior.

1.3- Pruebas y resultados.

En el cuarto capítulo se describe como se ha probado el diseño, tanto para comprobar los bloques como que el diseño funciona de la misma manera que uno comercial. Se ha comparado este diseño con el microcontrolador 8051 para comprobar cuanto se mejora tanto si hablamos de velocidad como en la reducción del tamaño de los programas.

Todos los bloques han sido comprobados en unos bancos de prueba mediante simulación con Modelsim. Por lo general cada bloque es diseñado y probado antes de pasar al siguiente. Pero conforme se ha ido avanzando en el proyecto se han requerido ampliar las funcionalidades de cada bloque y se han tenido que modificar. Tras estas modificaciones se han vuelto a comprobar que todo sigue funcionando bien. La mayoría de los bloques se han probado en bancos de pruebas independientes que se han creado de forma paralela al proyecto, sin embargo en las últimas etapas de diseño las pruebas

han sido conjuntas.

Las pruebas se hacen con unos vectores de pruebas denominados Testbench que generan unas señales que ponen a prueba las diferentes partes del diseño. Las pruebas de los últimos bloques se han hecho a base de código máquina con todos los bloques diseñados o con el diseño muy avanzado. Para poder probar el diseño se debe compilar antes, tarea que recae en una aplicación de Modelsim.

Para comprobar que nuestro diseño funciona igual que un microcontrolador comercial se ha probado haciendo funcionar un programa en C previamente compilado y simulado en Keil, un entorno de desarrollo que compila programas en C para la arquitectura del 251. Este programa genera un archivo en hexadecimal y realiza simulaciones del funcionamiento de estos microcontroladores. En este capítulo se pueden ver los resultados obtenidos al ejecutar un programa en C que multiplica dos matrices 3x3.

Una vez dado por finalizado el diseño se procede con las pruebas de síntesis y se mide el tamaño total que ocupa el hardware de este diseño. Para ello se usa Synplify. En realidad no solo se ha utilizado al final, sino que se han hecho pruebas frecuentemente ya que su compilador es más restrictivo que el de xst xilinx y detecta problemas de sintetizabilidad de manera más explícita. El resultado que nos proporciona nos permite ver el número de puertas equivalentes que ha resultado y la frecuencia del reloj máxima que podemos usar en el diseño para una determinada FPGA.

Por último se ha comparado el resultado de ejecutar un programa con Bytes, enteros y enteros largos (Char, Integer y Long Integer en C) en nuestro diseño y en el diseño del 51. Se ha usado el mismo programa de multiplicación de matrices pero extendiendo las variables de “int” a “long int” y “char”. Se pretende ver con esta prueba que la extensión a 16 o 32 bits acelera la ejecución de los programas y que su código ocupa menos espacio en la arquitectura del 251.

Capítulo 1: Introducción.

Capítulo 2:
Microcontrolador 80251.

A continuación vamos a analizar el microcontrolador 80251 de Intel. Haremos especial hincapié en los aspectos que más nos conciernen para el posterior diseño del núcleo del microprocesador.

Primero resaltar que esta familia de microcontroladores de 8 bits es una versión de alto rendimiento de su predecesora la familia de microcontroladores 8051. Se ha mejorado con esta nueva arquitectura el número de ciclos de reloj necesarios por ciclo de máquina que requerían sus antecesores y se ha aumentado el número de instrucciones el tamaño de los datos con los que trabaja y los modos de direccionamiento.

También se ha visto afectado el espacio de memoria. En la arquitectura del 51 se trabaja con una zona de código, otra zona externa para RAM y la zona de RAM interna. A estas zonas se accede de forma independiente con instrucciones dedicadas. Con la nueva arquitectura pasa a tener 4 zonas, las dos internas que hemos mencionado antes y otras dos externas. El acceso a estas zonas puede ser como antes, con instrucciones dedicadas, o accediendo con direccionamiento indirecto.

2.1- Características técnicas de la arquitectura 251(comercial).

- Bus de direcciones de 24 bit, que permiten acceder hasta 16MB de memoria.
- Bus de direcciones independiente entre la parte de memoria para datos de variables y datos de programa.
- Hasta 16 KB de memoria interna destinada a almacenar los programas ampliable a 64KB con memoria externa.
- 1KB de memoria RAM interna ampliable con memoria externa.
- Bancada de registros internos en la CPU accesibles como Bytes, Words y Double-words.
- 269 instrucciones que nos permiten operar con datos en formatos de 8, 16 y 32 bits.
- Reducción del número de ciclos de reloj de 12 ciclos del 51 a los 2 ciclos.
- Compatibilidad con programas realizados para la arquitectura 51 a través de la creación de dos modos de trabajo, binary y source.

Otras características técnicas de estos microcontroladores son la inclusión de una serie de periféricos. Entre ellos se encuentran Timers, Counters, watchdog....

2.2- Espacio de memoria.

Como ya se ha mencionado, la arquitectura del 251 posee un bus de datos interno de 24 bits, que permitiría disponer hasta 16MB de memoria. La memoria está dividida en varias zonas que pueden ser accedidas simultáneamente gracias a que existen varios buses conectados entre sí. Por ejemplo, la zona de código tiene un bus independiente que nos permite leer a la vez datos y código. El problema llega cuando se pretende leer datos en la memoria de código desde el programa. En este caso tan solo se puede realizar la lectura del dato deteniendo el Fetch.

El espacio de memoria está dividido en cinco zonas. Como dentro la familia 80251 hay diferentes modelos, para esta sección hemos usado el 83C251SB:

- **Memoria de código:** Localizada desde FF:0000 hasta FF:FFFF. Esta zona está reservada para código de programa y constantes. Este espacio está dividido en dos zonas, una interna y otra externa. La memoria interna es de 16 KB para nuestro modelo, el resto es externa.
- **Memoria externa 1:** Localizada desde FE:0000 hasta FE:FFFF. Esta zona de memoria está reservada para introducir bloques de memoria externa.
- **Memoria externa 2:** Localizada desde 01:0000 hasta 01:FFFF. Igual que antes, esta zona es para colocar memoria externa.

• **Memoria RAM interna:** Localizada desde 00:0000 hasta 00:FFFF. Igual que la memoria de código está dividida en dos zonas, una que contiene la RAM interna, que va desde 00:0000 hasta 00:1FFF y una segunda zona que puede ser completada con más memoria externa. La RAM interna es de 1024 Bytes.

Existe una quinta zona que va desde la 02:0000 hasta la FD:FFFF reservada que no es accesible por el usuario.

A continuación se muestra el mapa de memoria de arquitectura 251:

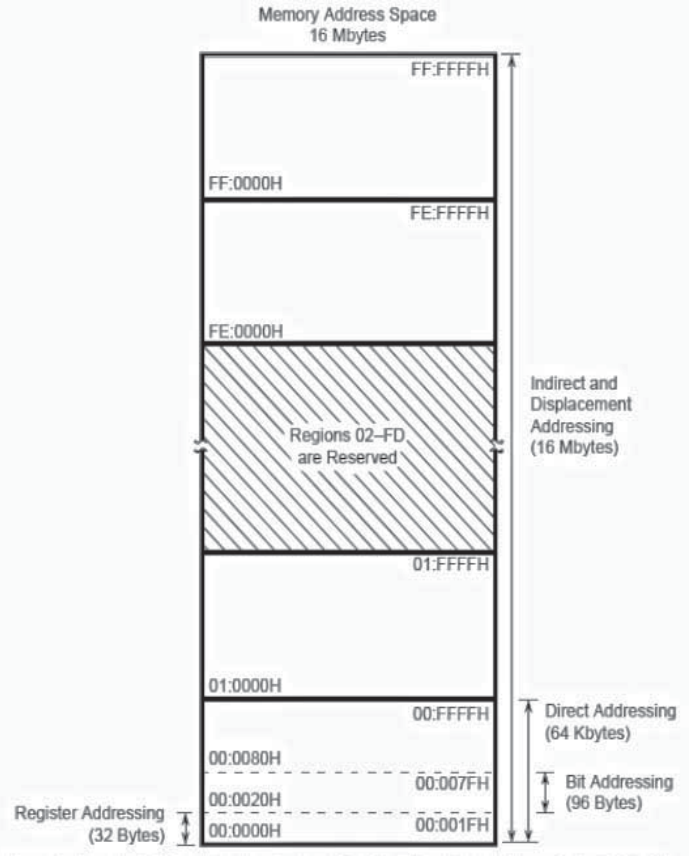


Figura 2.2_1: Espacio de memoria RAM y su direccionamiento.
Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

2.3 Modos de funcionamiento.

La arquitectura del 251 es compatible con sus predecesora, la 51. Uno de los aspectos más importantes para conseguir esta compatibilidad es poder reutilizar los programas realizados para la anterior tecnología, sin necesidad de recompilar el código.

Las instrucciones poseen tamaños variados, desde 1 Byte hasta 5 Bytes en el peor de los casos. Sin embargo, siempre con el primer Byte y en algunos casos con el segundo se sabe de qué instrucción se trata. El resto son datos y direcciones que no influyen en el tipo de instrucción. La arquitectura del 51 usa todos los Opcodes posibles en 8 bits salvo el A5, por lo que las nuevas instrucciones de la arquitectura del 251 no caben. La solución adoptada fue utilizar el Byte A5 como Byte de escape y dividir el espacio de instrucciones en tres partes. Una parte fija para ambas arquitecturas, y las otras con instrucciones propias de cada una de las arquitecturas.

De esta forma se adopto dos modos de trabajo, en el que el espacio de instrucciones cambia dependiendo del tipo de instrucciones que se pretendan usar. El primer modo se llama Binario (o Binary) y contiene todas las instrucciones del 51. Este modo se usa para reutilizar programas ya creados y no recompilarlos a la nueva tecnología. Para acceder a las instrucciones nuevas se debe poner el prefijo A5, que es el denominado Byte de escape. El segundo modo se denomina fuente (o Source) y agrupa todas las instrucciones nuevas de esta arquitectura. Estas instrucciones pueden trabajar con rangos mayores en direccionamiento directo e indirecto y con más registros. También aparecen nuevas instrucciones que realizan más operaciones. Al trabajar con datos más grandes y con más modos de direccionamiento es capaz de operar mucho más rápido y con menor código. Las instrucciones que se sustituyen en este modo se pueden usar añadiendo el prefijo A5.

La razón de que se decantasen por incluir dos modos de trabajo y no usar siempre el prefijo en las nuevas instrucciones es que en los microcontroladores comerciales leer un Opcode extra supone más ciclos de máquina por instrucción. Poniendo dos modos se consigue reducir el tiempo de ejecución de los programas, ya que pondremos el modo de trabajo de aquel que tengamos más instrucciones.

El problema antes mencionado de velocidad de acceso a las instrucciones se debe a que los microcontroladores 80251 funcionan con ciclos de máquina variables y la lectura del prefijo supone la adhesión de un ciclo de máquina más. En nuestro caso no supone ningún inconveniente y se podría trabajar directamente en el modo binario sin que suponga un tiempo de ejecución mayor. Es decir, trabajar en un modo u otro da el resultado en el mismo tiempo de ejecución.

2.3.1 Mapa de instrucciones.

Las instrucciones se dividen en tres bloques: Un bloque fijo y común, uno con instrucciones especificas del 51 y otro con instrucciones especificas del 251. La división se hace atendiendo a los Opcodes de las instrucciones, tomándose los Nibbles (4 Bits) de menor peso. El bloque común está compuesto por los Opcodes que van desde el “x0” hasta el “x7”, mientras que el segundo y el tercero van desde el “x8” hasta el “xF”. Según el modo de trabajo, al leer el Opcode interpretara una instrucción u otra.

A continuación se muestran unos ejemplos gráficos que ilustran el formato de las instrucciones y como se diferencian dependiendo del modo de trabajo:

Formato de las instrucciones por nibbles

7A	
Nibble 1 (mayor peso)	Nibble 2 (menor peso)

Ejemplos de instrucciones dependiendo del modo de trabajo

	Modo	
Opcode	**Source**	**Binary**
“42”	ORL A, dir8	ORL A, dir8
“8C”	DIV Rm, Rm	MOV dir8, Rn
“A5” “8C”	MOV dir8, Rn	DIV Rm, Rm

Figura 2.3.1_1: Formato de las instrucciones y ejemplos de instrucciones trabajando en modo binario y fuente.

Fuente: Elaboración propia.

Las siguientes figuras muestran el mapa de instrucciones dependiendo del modo de trabajo.

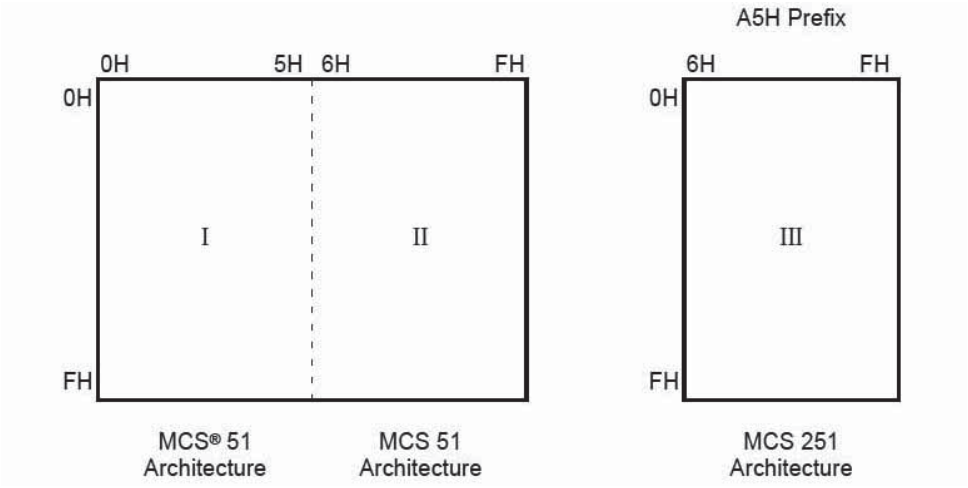


Figura 2.3.1_2: Mapa de memoria trabajando en modo binario.
Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

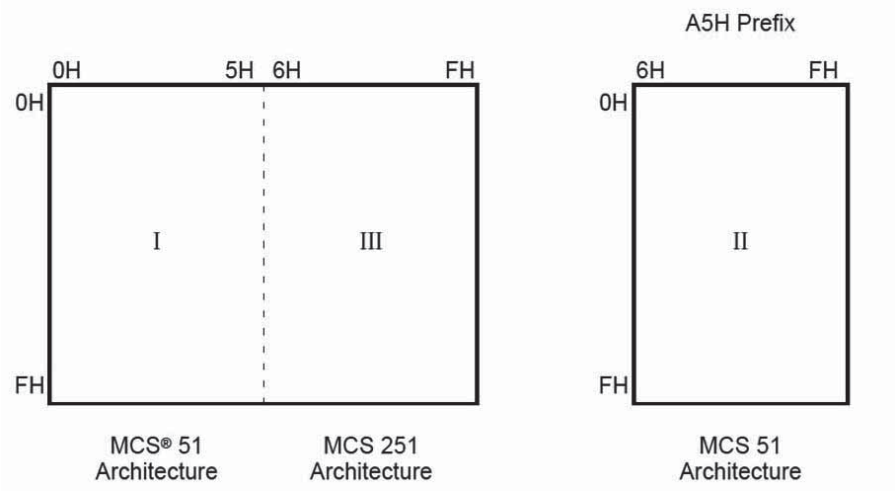


Figura 2.3.1_3: Mapa de memoria trabajando en modo fuente.
Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

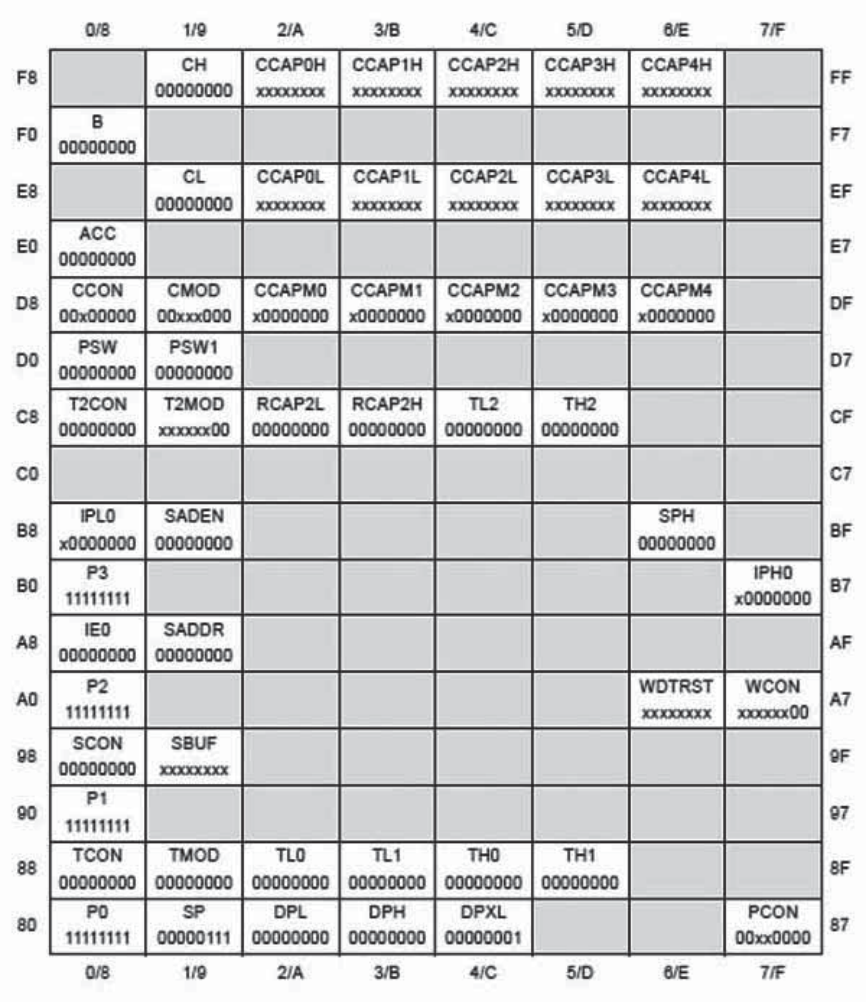
La primera figura se corresponde con el funcionamiento en modo binario y el segundo funciona en modo fuente. Observamos como dependiendo del modo de trabajo debemos introducir el prefijo A5 para las instrucciones del otro modo.

2.4 Registros internos.

Además de las memorias RAM/ROM interna, los microcontroladores 80251 cuentan con un banco de registros y los SFR´s. La diferencia entre los bloques RAM y los registros son que los registros están formados por biestables por lo que se puede acceder simultáneamente a varios registros. Otra característica es que al efectuarse un Reset, estos registros se ponen a cero, mientras que la RAM mantiene los datos.

Podemos clasificar los registros en tres tipos: Special Function Registers (SFR), banco de registros y registros especiales.

2.4.1 Registros Special Function Registers (SFR):



	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
F8		CH 00000000	CCAP0H xxxxxxxx	CCAP1H xxxxxxxx	CCAP2H xxxxxxxx	CCAP3H xxxxxxxx	CCAP4H xxxxxxxx		FF
F0	B 00000000								F7
E8		CL 00000000	CCAP0L xxxxxxxx	CCAP1L xxxxxxxx	CCAP2L xxxxxxxx	CCAP3L xxxxxxxx	CCAP4L xxxxxxxx		EF
E0	ACC 00000000								E7
D8	CCON 00x00000	CMOD 00xxx000	CCAPM0 x0000000	CCAPM1 x0000000	CCAPM2 x0000000	CCAPM3 x0000000	CCAPM4 x0000000		DF
D0	PSW 00000000	PSW1 00000000							D7
C8	T2CON 00000000	T2MOD xxxxxx00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			CF
C0									C7
B8	IPL0 x0000000	SADEN 00000000					SPH 00000000		BF
B0	P3 11111111							IPH0 x0000000	B7
A8	IE0 00000000	SADDR 00000000							AF
A0	P2 11111111						WDTRST xxxxxxxx	WCON xxxxxx00	A7
98	SCON 00000000	SBUF xxxxxxxx							9F
90	P1 11111111								97
88	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000			8F
80	P0 11111111	SP 00000111	DPL 00000000	DPH 00000000	DPXL 00000001			PCON 00xx0000	87
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Figura 2.4.1_1: Espacio de memoria para los Special Function Register (SFR).

Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

Estos registros incluyen los periféricos del microcontrolador, los puertos, las interrupciones, el acumulador... No todo el espacio reservado es accesible, tan solo una parte está ocupada. A estos registros se accede con direccionamiento directo como si fuera RAM. Las direcciones comprendidas entre “80H” hasta “FFH” están reservadas para estos registros. La figura 2.4.1_1 muestra los registros existentes junto con su nombre y la dirección RAM donde se encuentran.

2.4.2 Banco de registros:

La estructura del banco de registros es bastante compleja. Para empezar, los registros pueden ser direccionados en forma de Byte(R), Word (W) u Double-word (DW).

Aunque se direccionen datos más grandes, los registros englobados son los mismos, pero es capaz de acceder a más registros. En otras palabras, hay un total de 16 registros que son accesibles con una instrucción tipo Byte. Esos 16 registros pueden ser accesibles con una instrucción de Word, pero de dos en dos. Además podemos acceder a un total de 32 Bytes con este direccionamiento. Con los Double-word pasa lo mismo, podemos acceder a los mismos 32 Bytes que con el direccionamiento Word o a los 16 del tipo Byte, pero en este caso se amplía 8 Bytes más. El direccionamiento de estos 8

Bytes se corresponde a la parte alta de las direcciones. En estos casos los datos están siempre alineados, empezando por cero o por valores pares que van de dos en dos o de cuatro en cuatro. En la figura 2.4.2_1 se puede observar el banco de registros y para cada tipo de formato los Bytes que engloba. Los registros que no aparecen en el recuadro de cada tipo de direccionamiento son porque no son accesibles.

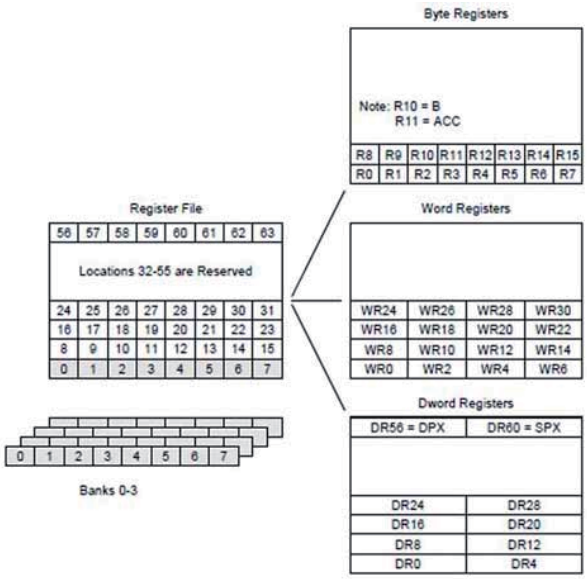


Figura 2.4.2_1: Banco de registros y su nomenclatura dependiendo del formato como se direcciona.

Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

Se accede a los registros a partir de cuatro bits del Opcode de la instrucción. Por lo general, la parte alta del Opcode suele ser la dirección y la parte baja es el tamaño del dato.

Aunque con Bytes solo se pueden acceder a los 16 primeros registros, la nomenclatura del resto de direccionamientos parte del valor del primer registro que tendría si se direccionara como Byte. Por ejemplo, el W18 contiene los registros R18 y R19 registros que no pueden ser accedidos como Bytes.

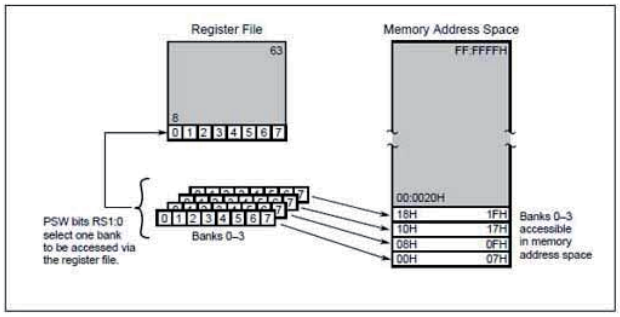
Los registros que van desde R0 hasta R7 (W0-W6 o DW0 y DW4) son registros que se despliegan lateralmente en 4 filas haciendo un total de 32 Bytes. Se controlan con los Bits RS de los registros PSW0 o PSW1. Dependiendo del valor de RS que va desde “00” hasta “11” seleccionamos una de las 4 filas. Cuando direccionamos un registro comprendido en el rango antes mencionado se accede directamente a una de estas filas.

Estos 32 Bytes tienen otra peculiaridad, y es que además de funcionar como registros lo hacen como la parte baja de la memoria. Tanto en direccionamiento directo como indirecto si pretendemos leer o escribir cualquier dirección entre la “00H” y la “1FH” lo hace en estos registros. El despliegue se hace siguiendo el orden ascendente de RS, por lo que la fila de RS = “00b”corresponde con las direcciones de la “00H” a la “08H”, mientras que la dirección RS = “11b” se corresponden desde la “19H” a la “1FH”.

Como forma parte de la RAM, el acceso a estos registros no es alineado. Podemos acceder a cualquier registro independientemente del tipo de dato sin necesidad que sea par.

En nuestro diseño se ha ampliado el banco de registros de 64 a 84 registros, eliminando el vacío que hay entre el registro 32 hasta el 55.

En la figura 2.4.2_2 Vemos como se despliega el banco de registro para formar la parte baja de la memoria RAM. La dirección de la izquierda representa el registro PSW que contiene el vector RS. La dirección de la derecha se corresponde a la memoria RAM.



Register File Locations 0–7

Register Bank Selection

Bank	Address Range	PSW Selection Bits	
		RS1	RS0
Bank 0	00H–07H	0	0
Bank 1	08H–0FH	0	1
Bank 2	10H–17H	1	0
Bank 3	18H–1FH	1	1

Figura 2.4.2_2: Registros desplegables que funcionan como la parte baja de la memoria RAM. Se puede observar la asignación del espacio de memoria para cada registro.
Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

2.4.3 Registros especiales:

Estos registros en realidad forman parte de los dos grupos anteriores a la vez. En otras palabras, se pueden acceder a ellos desde el banco de registros o desde los registros SFR. Por lo general tienen funciones especiales dentro de las instrucciones. Se corresponden con el DPTX, SPX y los acumuladores. Por orden de mayor a menor, primero nos encontraríamos el SPX, con 2 Bytes y se corresponde a DR60, el DPTX con 3 Bytes y dirección DR56, y por último los acumuladores de un Byte A y B con direcciones R11 y R10 respectivamente. Los registros SPX y DPTS son punteros que direccionan en el primer caso la dirección de la pila y en el segundo forma la parte alta de la dirección cuando se accede a memoria externa. En la pila que antes hemos comentado es una parte de la memoria en donde se almacenan datos importantes cuando se realizan subrutinas o interrupciones, como es el caso del PC (Program Counter) o del PSW1. La figura 2.4.3_1 se observa los registros en el banco de registros y en espacio de memoria con sus respectivas direcciones en ambos espacios.

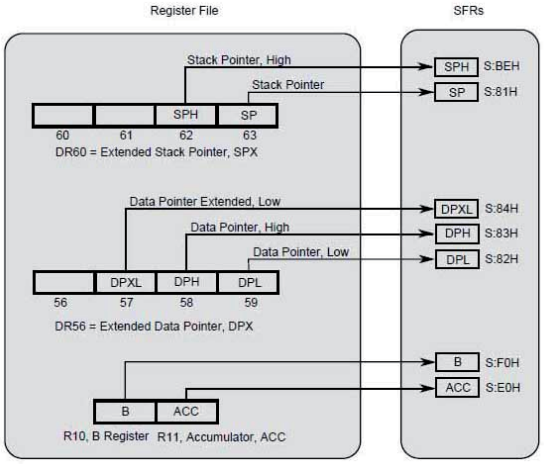


Figura 2.4.3_1: Registros compartidos entre el banco de registros y el banco SFR. Se indican las direcciones en ambos bancos.
Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

Otros registros especiales son los PSW (Program Status Word) que si bien no son accesibles desde el banco de registros, estos registros se ven modificados al ejecutar un gran número de instrucciones. Casi todas instrucciones modifican algunos bits de estos registros sin necesidad de ser direccionados. Cada bit tiene funcionas propias y se denominan Flags. En la figura 2.4.3_2 se aprecia ambos registros y cuál es la función de cada Flag.

PSW Address: S:D0H Reset State: 0000 0000B

7							0
CY	AC	F0	RS1	RS0	OV	UD	P

Bit Number	Bit Mnemonic	Function
7	CY	Carry Flag: The carry flag is set by an addition instruction (ADD, ADDC) if there is a carry out of the MSB. It is set by a subtraction (SUB, SUBB) or compare (CMP) if a borrow is needed for the MSB. The carry flag is also affected by some rotate and shift instructions, logical bit instructions, bit move instructions, and the multiply (MUL) and decimal adjust (DA) instructions (see Table 5-10).
6	AC	Auxiliary Carry Flag: The auxiliary carry flag is affected only by instructions that address 8-bit operands. The AC flag is set if an arithmetic instruction with an 8-bit operand produces a carry out of bit 3 (from addition) or a borrow into bit 3 (from subtraction). Otherwise, it is cleared. This flag is useful for BCD arithmetic (see Table 5-10).
5	F0	Flag 0: This general-purpose flag is available to the user.
4:3	RS1:0	Register Bank Select Bits 1 and 0: These bits select the memory locations that comprise the active bank of the register file (registers R0–R7). RS1 RS0 Bank Address 0 0 0 00H–07H 0 1 1 08H–0FH 1 0 2 10H–17H 1 1 3 18H–1FH
2	OV	Overflow Flag: This bit is set if an addition or subtraction of signed variables results in an overflow error (i.e., if the magnitude of the sum or difference is too great for the seven LSBs in 2's-complement representation). The overflow flag is also set if a multiplication product overflows one byte or if a division by zero is attempted.
1	UD	User-definable Flag: This general-purpose flag is available to the user.
0	P	Parity Bit: This bit indicates the parity of the accumulator. It is set if an odd number of bits in the accumulator are set. Otherwise, it is cleared. Not all instructions update the parity bit. The parity bit is set or cleared by instructions that change the contents of the accumulator (ACC, Register R11).

PSW1 Address: S:D1H Reset State: 0000 0000B

7							0
CY	AC	N	RS1	RS0	OV	Z	—

Bit Number	Bit Mnemonic	Function
7	CY	Carry Flag: Identical to the CY bit in the PSW register (Figure 5-2).
6	AC	Auxiliary Carry Flag: Identical to the AC bit in the PSW register (Figure 5-2).
5	N	Negative Flag: This bit is set if the result of the last logical or arithmetic operation was negative (i.e., bit 15 = 1). Otherwise it is cleared.
4–3	RS1:0	Register Bank Select Bits 0 and 1: Identical to the RS1:0 bits in the PSW register (Figure 5-2).
2	OV	Overflow Flag: Identical to the OV bit in the PSW register (Figure 5-2).
1	Z	Zero Flag: This flag is set if the result of the last logical or arithmetic operation is zero. Otherwise it is cleared.
0	—	Reserved: The value read from this bit is indeterminate. Write a "0" to this bit.

Figura 2.4.3_2: Registros PSW0 y PSW1 junto con la definición de todos sus Flags.
Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

2.5 Modos de direccionamiento.

La arquitectura del 251 posee gran variedad de modos de direccionamiento por los que una instrucción puede obtener los datos necesarios para operar con ellos. Además de leer datos de la RAM o de los registros existen otras formas de obtener estos datos:

- **Direccionamiento de registros:** La instrucción contiene el registro que se pretende leer. La dirección es de 4 bits y puede direccionar Bytes, Words o Double-words como se ha comentado en la sección 2.4.

- **Direccionamiento directo:** La instrucción contiene la dirección de la memoria, pudiendo estar en 1 Byte o en 2 Bytes (8 o 16 bits). Con este direccionamiento se accede a direcciones fijas pero debemos tener cuidado con los SFR que se encuentran entre las direcciones “80H” y “FFH”. En la sección 2.5.1 se amplía la información.

- **Direccionamiento indirecto:** En este caso la dirección se encuentra en un registro. Para acceder al dato se lee primero este registro que es procesado para obtener la dirección real. En la sección 2.5.2 se amplía la información.

- **Direccionamiento inmediato:** En este caso el dato no está ni en memoria RAM ni en registros, sino que la instrucción contiene el dato. Puede ser de 8 bits o de 16 bits.

- **Direccionamiento con desplazamiento:** Caso especial del indirecto, la instrucción contiene el registro que contiene la dirección de RAM y a la vez un offset que será sumado a la dirección. El resultado es la dirección donde está el dato. El offset puede ser de 16 o 24 bits.

- **Direccionamiento relativo:** Similar al direccionamiento con desplazamiento pero este hace referencia a instrucciones de salto. El resultado de la suma del valor del PC y el offset da una dirección de programa al que el PC debe saltar.

- **Direccionamiento de Bits:** Hay dos tipos de direccionamiento de bits. Pero en ambas se obtiene una dirección de un Byte y la posición del bit. En la sección 2.5.3 se amplía la información.

El resultado de las operaciones, cuando hay que escribirlo, se realiza direccionando las memorias internas o registros. No tienen tantos modos de direccionamiento como con la lectura de datos.

Sin embargo en la lista anterior queda por explicar con más de profundidad algunos tipos de direccionamiento de las memorias RAM.

2.5.1 Direccionamiento directo.

La dirección a la que accede está dentro de la instrucción, direccionando posiciones fijas de variables. Las direcciones son de 8 o 16 bits aunque la extensión de las memorias de los dispositivos comerciales solo alcanzan hasta los 10 bits de direcciones. En nuestro caso la memoria abarca los 16 bits de direcciones pues usamos memorias

internas más grandes. Debemos tener cuidado con las direcciones comprendidas entre “80H” y “FFH” que se corresponden a los SFR. Aunque sean registros, se acceden mediante direccionamiento directo. La memoria RAM que hay detrás solo se puede acceder con direccionamiento indirecto.

El rango de acceso con este direccionamiento va desde la dirección “0000H” hasta la “FFFFH”. Solo depende si usamos 8 o 16 bits.

2.5.2 Direccionamiento indirecto.

La dirección a la que se desea acceder esta dentro de un registro. Los registros pueden tener el tamaño de un Byte compatible con la arquitectura del 51, de 2 Byte o un Word, que nos permite acceder a la primera zona de memoria, y por último 4Bytes o Double-word, dándonos acceso a los 16Mb. Los dos últimos tan solo permiten leer o escribir dos Bytes. La instrucción contiene la dirección del registro donde se almacena la dirección de la memoria que se desea acceder. En algunas instrucciones a la dirección leída del registro se le suma un valor fijo.

El rango de acceso de este caso va de “00H” hasta la “FFH” con direccionamiento indirecto compatible con el 51. De la dirección “0000H” hasta la “FFFFH” si usamos Word como base de la dirección. Y si elegimos Double-word podemos acceder desde “00:0000H” hasta la dirección “FF:FFFFH”.

2.5.3 Direccionamiento de bits.

Este direccionamiento obtiene de un Byte determinado el valor de un bit. Por lo tanto se necesita dos direcciones, la del Byte y la del bit. Se sigue dos estrategias, dependiendo de la tecnología de la arquitectura.

La versión del 251 sigue una estrategia muy simple, ya que la instrucción contiene la dirección del dato y la posición del bit a leer.

Sin embargo la versión del 51 es más complicada, ya que ambas direcciones se encuentran en un Byte y dependiendo del valor de este accede a RAM o a los SFR. La primera mitad accede a los 16 primeros Bytes y la segunda mitad se mueve por los registros SFR que terminan en cero u ocho: S:80H, S:88H, S:90H….

Podemos ver el mapa de direcciones del direccionamiento bit51 en la tabla1:

	Direccionamiento de bits según formato del 51. (bit51)																											
	Direcciones parte baja de RAM: 00:0020 – 00:002F																											
	BYTE 3								BYTE 2								BYTE 1								BYTE 0			
00:002F-2C	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61	60
00:002B-28	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41	40
00:0027-24	3F	3E	3D	3C	3B	3A	39	38	37	36	35	34	33	32	31	30	2F	2E	2D	2C	2B	2A	29	28	27	26	25	24	23	22	21	20
00:0023-20	1F	1E	1D	1C	1B	1A	19	18	17	16	15	14	13	12	11	10	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Direcciones de registros SFR: 00:0080 – 00:00F8																	
Registro SFR	Valor instrucción								Registro SFR	Valor instrucción							
00:0080	87	86	85	84	83	82	81	80	00:00C0	C7	C6	C5	C4	C3	C2	C1	C0
00:0088	8F	8E	8D	8C	8B	8A	89	88	00:00C8	CF	CE	CD	CC	CB	CA	C9	C8
00:0090	97	96	95	94	93	92	91	90	00:00D0	D7	D6	D5	D4	D3	D2	D1	D0
00:0098	9F	9E	9D	9C	9B	9A	99	98	00:00D8	DF	DE	DD	DC	DB	DA	D9	D8
00:00A0	A7	A6	A5	A4	A3	A2	A1	A0	00:00E0	E7	E6	E5	E4	E3	E2	E1	E0
00:00A8	AF	AE	AD	AC	AB	AA	A9	A8	00:00E8	EF	EE	DE	EC	EB	EA	E9	E8
00:00B0	B7	B6	B5	B4	B3	B2	B1	B0	00:00F0	F7	F6	F5	F4	F3	F2	F1	F0
00:00B8	BF	BE	BD	BC	BB	BA	B9	B8	00:00F8	FF	FE	FD	FC	FB	FA	F9	F8

Tabla 1: Bits direccionados con el formato compatible del 51 (bit51). Dependiendo de la dirección se accede hasta la dirección 2F de la memoria RAM o a ciertas direcciones de los registros SFR.
Fuente: Elaboración propia.

2.6 Programación.

En esta sección vamos a hablar de forma general sobre una serie de consideraciones a tener en cuenta a la hora de programar los microcontroladores 80251.

2.6.1 Formato de los Datos.

Como ya se ha comentado, los microcontroladores con arquitectura 251 pueden trabajar con datos de 8, 16 y 32 bits.

Tipo de dato	Número de Bits
Bit	1
Byte	8
Word	16
Double word (Dword)	32

Tabla 2: Número de bits de los formatos de los datos.
Fuente: Elaboración propia.

En caso da leer o escribir un dato con más de un Byte se debe tener en cuenta que estos se guardan en formato Big-endian.

Big-endian es un formato de almacenamiento en el que los Bytes de más peso se sitúan en las posiciones más bajas. El Byte más significativo (MSB) del Word o Double-Word se almacenará, por lo tanto, en la dirección especificada por la instrucción. El resto de Bytes se almacenan direcciones mayores hasta llegar al Byte menos significativo (LSB) que es almacenado en la dirección más alta. En la figura 2.6.1_1 se muestra 3 ejemplos de cómo funciona este método.

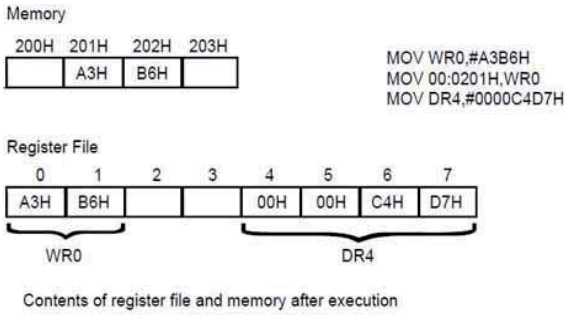


Figura 2.6.1_1: Orden de almacenamiento de datos en formato Big-endian.
Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

2.6.2 Notaciones de los datos.

Dependiendo del tamaño de los datos o de la arquitectura en la que se introdujeron los datos, así como el tipo de instrucción, se denotan de forma distinta.

En la tabla 3 se recoge las notaciones de los datos dependiendo de la arquitectura y del formato del dato, así como el rango de valores y para la arquitectura 251 diferencia entre datos destino y datos fuente.

Notación de los registros según tamaño del dato y arquitectura utilizada.

Arquitectura	Tipo de registro	Símbolo	Rango	Registro destino	Registro Fuente
51	Byte	Ri	R0, R1	---	---
51		Rn	R0-R7	---	---
251		Rm	R0-R15	Rmd	Rms
251	Word	WRj	WR0-WR30	WRjd	WRjs
251	Dword	DRk	DR0-DR60	DRkd	DRks

Tabla 3: Notación de los registros dependiendo del tamaño y de la arquitectura.
Fuente: Elaboración propia.

2.6.3 Clasificación de las instrucciones.

La arquitectura del 251 tiene 269 instrucciones. La mayoría se pueden clasificar por las operaciones que desempeñan.

- **Instrucciones aritméticas:** Los microcontroladores 80251 poseen una amplia gama de operaciones aritméticas. La mayoría soporta todos los formatos de tamaño de datos, pero algunas operaciones solo pueden operar con datos de hasta 16bits. Todas las operaciones aritméticas modifican el PSW de alguna manera. Algunas operaciones pueden operar con datos inmediatos e incluso algunas operaciones se realizan teniendo en cuenta el bit Carry del PSW. En la tabla siguiente se enumeran las instrucciones aritméticas así como los operadores con los que puede trabajar.

Operación aritmética	Notación	Dato inmediato	32 Bits	Carry*1	Lectura Ram*2	Escritura en Ram	Opera con el acumulador Acc*3	Tamaño del resultado*4
Suma	ADD	si	si	no	si	no	si	=
	ADDC	si	no	si	si	no	si	=
Resta	SUB	si	si	no	si	no	no	=
	SUBB	si	no	si	si	no	si	=
Comparación	CMP	si	si	no	si	no	no	0
Incremento	INC	no	si	no	no	si	si	=
Decremento	DEC	no	si	no	no	si	si	=
Multiplicación	MUL	no	no	no	no	no	si	x2
División	DIV	no	no	no	no	no	si	x2

*1 Carry: Se considera solamente cuando se usa como operador, no como Flag.
*2 Lectura de la Ram, para usarla como operador. Si luego se guarda en ella no cuenta.
*3 Opera con el acumulador ACC: Si uno de los operadores es el acumulador, tanto en la lectura como escritura.
*4 Tamaño del resultado: '=' mismo tamaño que los operadores, '0' si resultado no se guarda, 'x2' si el resultado ocupa el doble que los operandos

Tabla 4: Instrucciones aritméticas.

Fuente: Elaboración propia.

• **Instrucciones lógicas:** Estas instrucciones realizan diferentes operaciones lógicas. La tabla siguiente muestra los tipos de instrucciones disponibles.

Instrucción lógica	Notación	Descripción	Ejemplo
AND lógica	ANL	Realiza la operación lógica AND.	“0101”&”1100”=”0100”
OR lógica	ORL	Realiza la operación lógica OR.	“0101”V”1100”=”1101”
OR exclusivo	XRL	Realiza la operación lógica OR exclusivo.	“0101”xor”1100”=”0011”
Reinicio	CLR	Pone el acumulador a cero.	A=>00H
Complemento	CPL	Invierte el dato. Donde había un cero pone un 1 y viceversa	“10001011”=>”01110100”
Rotaciones	RR	Rota los bit hacia la derecha. Los bits salientes entran por la izquierda.	“100111”=>”001111”
	RL	Rota los bit hacia la izquierda. Los bits salientes entran por la derecha.	“01101101”=>”10110110”
	RRC	Rota los bits hacia la derecha, y los bits salientes van al CY.	Cy=0, “1001”=>CY=1, “0010”
	RLC	Rota los bits hacia la izquierda, y los bits salientes van al CY.	CY =1 “00101110”=> CY = 0, “10010111”
Desplazamiento	SLL	Los desplazamientos mueven los datos hacia la izquienda(SLI) o hacia la derecha(SRx). Si los desplaza hacia la izquierda por la derecha introduce ceros. Hacia la derecha introduce ceros si SRL o bits de signo si SRA.	“11011001”=>”10110010”
	SRA		“10010011”=>”11001001”
	SRL		“10001011”=>”01000101”

Tabla 5: Instrucciones lógicas.

Fuente: Elaboración propia.

• **Instrucciones de transferencia de datos:** Estas instrucciones leen un dato y lo escriben en otra posición. Son las más numerosas que hay y son posibles todas las combinaciones de direccionamiento con todo tipo de tamaños de datos. A grandes rasgos, las instrucciones pueden ser:

◦ ***Instrucciones MOV:*** La instrucción más numerosa con diferencia, puede funcionar con todos los formatos de datos y usa todos los direccionamientos salvo el relativo. Esta instrucción lee un dato que puede estar en RAM o en un registro y lo escribe donde se le mande, ya sea RAM o un registro.

- ***Instrucciones MOVX y MOVC:*** Estos casos especiales de la instrucción MOV leen memoria externa (MOVX) o memoria de código (MOVC) y lo guardan en el acumulador.

- ***Instrucciones MOVS y MOVZ:*** Toman un dato de 8 bits y lo guardan en una posición de 16 bits, completando el Byte que falta con ceros (MOVZ) o con bit de signo (MOVS).

- ***Instrucciones de cambio (XCH):*** Leen el acumulador y otra dirección de memoria RAM. Los datos son intercambiados, escribiendo el valor del acumulador en la posición de memoria y viceversa. La instrucción XCHD intercambia el Nibble bajo del acumulador con el Nibble bajo de un dato de la RAM.

- ***Instrucciones POP y PUSH:*** Permiten guardar y luego recuperar información. Push guarda un dato en la pila y la instrucción POP la recupera. Ambos pueden operar con la RAM o con registros.

• **Instrucciones de Bits:** Leen un bit de una determinada instrucción y operan con él. Lo más normal es que afecten al Flag Carry. Hay cuatro categorías de estas instrucciones como muestra la tabla 2.5.2_3.

Categoría	Operación	Instrucciones	DESCRIPCIÓN
#1	absolutas	SETB, CLR, CPL	Ponen a '1', '0' o al valor inverso a un determinado Bit.
#2	Lógicas	ANL, ANL/, ORL, ORL/	Realizan la operación lógica AND u OR. Pueden tomar directamente el bit o su inverso
#3	Copiado	MOV	Copian un bit de un byte en el CY o viceversa.
#4	Salto	JNB, JB, JNB	Se produce un salto en el programa dependiendo del valor del bit.

Tabla 6: Instrucciones de bit.

Fuente: Elaboración propia.

• **Instrucciones de salto condicional:** Realizan cambios de programa si se da cierta condición. Antes de utilizar esta instrucción, se debe realizar una instrucción de comparación o aritmética. Suelen usar el registro PSW1 para tomar los bits que determinaran si la condición de salto se da, pero algunas instrucciones leen un bit en RAM para determinarla. En caso positivo, la dirección de salto se toma sumando al PC un cierto valor fijo en la instrucción. El rango de salto siempre está en los 128 alrededor de su posición. A continuación se muestra una tabla con las instrucciones de salto según el resultado deseado tras la comparación.

Operand Type	Relation					
	=	≠	>	<	≥	≤
Unsigned	JE	JNE	JG	JL	JGE	JLE
Signed			JSG	JSL	JSGE	JSLE

Tabla 7: Instrucciones de salto condicional junto con la condición de salto.

Fuente: 8X251SA, 8X251SB, 8X251SP, 8X251SQ Embedded Microcontroller User´s Manual.

• **Instrucciones de salto absoluto:** Estas instrucciones producen un cambio de programa poniendo el PC en una determinada dirección. Hay cinco tipos de saltos absolutos, que son definidos en la tabla 2.5.4_5.

Instrucción	Rango del salto
NOP	1 byte
SJMP	256 bytes (+127,-128)
AJMP	FF:0000H – FF:03FFH
LJMP	FF:0000H – FF:FFFFH
EJMP	00:0000H – FF:FFFFH

Tabla 8: Instrucciones de salto absoluto.

Fuente: Elaboración propia.

• **Instrucciones de llamada y retorno:** Las instrucciones de llamada (CALL) son iguales que las instrucciones de salto pero estas almacenan en la pila la dirección de la siguiente instrucción. Se usa cuando el programa debe ejecutar una subrutina para luego continuar con el programa principal. El retorno se hace con las instrucciones RET o RETI, que leen el último valor de la pila y saltan a esa posición. En la tabla 2.5.4_6 vemos las instrucciones disponibles.

Instrucción	Rango del salto	Guarda PC / Nº bytes	Carga Pc / Nº bytes
ACALL	11 bits	Si / 2 bytes	no
LCALL	16 bits	Si / 2 bytes	no
ECALL	24 bits	Si / 24 bytes	no
RET	16 bits	no	Si / 2 bytes
RETI	24 bits	no	Si / 3 bytes + psw1

Tabla 9: Instrucciones de llamada y retorno.

Fuente: Elaboración propia.

Capítulo 3: Diseño del núcleo del microprocesador 80251.

Tras el estudio del microcontrolador 80251 se procede al diseño del núcleo del microprocesador. Pero antes del diseño propiamente dicho, se debe planificar una serie de detalles para conseguir unos objetivos de diseño. El principal objetivo de diseño de nuestro micro es que el tiempo de ciclo de ejecución de las instrucciones sea el mínimo.

Para lograr el objetivo de tiempo mínimo se han tenido en consideración varias alternativas en los puntos de diseño más importantes, como son las fases básicas del microprocesador o como el acceso a las memorias.

El diseño del microprocesador se ha realizado en VHDL con ayuda de un programa para realizar esquemáticos. El diseño de los esquemáticos se ha realizado de forma jerárquica y parte de la placa donde se encontraría el microcontrolador con el resto de componentes. El siguiente nivel se corresponde al microcontrolador, en donde se sitúa el núcleo con los periféricos. En nuestro caso, solo nos encontramos el núcleo y las memorias. El tercer nivel es el núcleo del microcontrolador. El núcleo se ha dividido en varios bloques según los diferentes procesos que deben ser desarrollados. Cada bloque tiene un esquemático, pero en estos casos están en el mismo nivel.

3.1 Procesos básicos del microprocesador.

El funcionamiento de un microprocesador se divide en cinco procesos que se realizan secuencialmente. Estos cinco procesos son: Fetch, Decode, Read, Execute y Write.

El primer proceso es el Fetch y es el encargado de leer el código en una dirección determinada de la memoria mediante un puntero llamado PC (Program Counter). Los datos leídos se almacenan en una pila interna y conforme se va desarrollando la ejecución de los programas, suministra al siguiente proceso estos datos. También se encarga de la lectura de esta memoria cuando una instrucción necesita un dato que está en código.

El siguiente proceso es el Decode. Decodifica las instrucciones que le llegan del Fetch y genera una serie de señales de control. Casi todas las señales de control se generan con este proceso, incluidas las de lectura o escritura de las memorias y el control del Execute.

El proceso Read recibe del anterior la dirección y el formato de dato y lee de la memoria o del banco de registros. Los datos los envía directamente al proceso Execute.

El proceso Execute es el encargado de realizar las operaciones de las instrucciones. Los datos llegan en su mayoría del proceso Read y la operación a realizar del Decode.

El último proceso es el Write, escribe en RAM o registros el resultado del Execute. Es casi igual al proceso Read.

Como se puede observar, cada proceso genera información esencial para el siguiente proceso. Hasta que el proceso no ha finalizado no se puede continuar con el siguiente. Aunque se pueden seguir estrategias para acelerar el proceso de ejecución de las instrucciones. Las más comunes son el desarrollo en paralelo de estos procesos. Otras formas son aumentar los buses de lectura de memorias, tanto RAM como de código.

Antes de comenzar nuestro diseño tuvimos muy en cuenta la estrategia que debíamos seguir para conseguir la mayor velocidad en la ejecución de las instrucciones. Nos planteamos diferentes estrategias para conseguir minimizar el tiempo de ejecución de una instrucción. También tuvimos en consideración el tamaño del bus de lectura de las memorias para poder realizar las lecturas y escrituras a tiempo.

3.1.1 Opción 1: Modo secuencial.

Ciclo 1	Ciclo 2	Ciclo 3	Ciclo4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9	Ciclo 10
Fetch	Decode	Read	Execute	Write	Fetch	Decode	Read	Execute	Write

Tabla 10: Modo de trabajo secuencial con cinco ciclos de reloj por ciclo de máquina.
Fuente: Elaboración propia.

La forma básica de desarrollo. Cada proceso se realiza al finalizar el anterior. Este método es sencillo y requiere muy poco hardware. Sin embargo es muy lento al requerir siempre al menos 5 ciclos para cada instrucción. Presenta el problema de la lectura de código por lo que debemos seleccionar una memoria que permita leer los Opcodes que

necesitamos. La solución optima es realizar por instrucción dos Fetch en una memoria de 32 bits por el tema de cuando salte no tengamos problemas con el Fetch.

3.1.2 Opción 2: Modo Pipeline.

Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9
Fetch 1	Fetch 2	Fetch 3	Fetch 4	Fetch 5	Fetch 6	Fetch 7	Fetch 8	Fetch 9
	Decode 1	Decode 2	Decode 3	Decode 4	Decode 5	Decode 6	Decode 7	Decode 8
		Read 1	Read 2	Read 3	Read 4	Read 5	Read 6	Read 7
			Execute 1	Execute 2	Execute 3	Execute 4	Execute 5	Execute 6
				Write 1	Write 2	Write 3	Write 4	Write 5

Tabla 11: Modo de trabajo paralelo. Las instrucciones solo tardan un ciclo de reloj en completarse.

Fuente: Elaboración propia.

En esta opción se realizan todas las tareas a la vez, pero de diferentes instrucciones. Esta opción reduce el tiempo de ejecución de una instrucción a tan solo un ciclo de máquina, y si el acceso a memoria está bien dimensionado se consigue que el ciclo máquina sea de tan solo un ciclo de reloj. Por contra la complejidad de esta forma es muy alta. Además el resultado de una instrucción puede ser requerida antes de que sea escrita, teniendo que interrumpir la ejecución de la instrucción. También se puede dar el caso de que el sistema pretenda leer y escribir en la RAM a la vez. La figura 2.1.2 muestra esta estrategia funcionando sin pausas.

3.1.3 Opción 3: Modo mixto.

Ciclo 1		Ciclo 2		Ciclo 3		Ciclo 4	
Fetch 1.1	Fetch 1.2	Fetch 2		Fetch 3		Fetch 4	
		Decode 1	Execute 1	Decode 2	Execute 2	Decode 3	Execute 3
		Read 1	Write 1	Read 2	Write 2	Read 2	Write 3

Tabla 12: Modo de trabajo mixto. Es el que finalmente utilizamos en nuestro diseño.

Fuente: Elaboración propia.

Es una mezcla de los dos anteriores. Por un lado es más rápido que el modo secuencial, por el otro, es más sencillo que el modo paralelo. No hay que interrumpir la ejecución de las instrucciones, pero en algunos casos si la ejecución del programa. La configuración de las memorias en este caso adquiere especial relevancia. La tabla 12 muestra este modo de funcionamiento.

Se realiza por un lado el Fetch de las instrucciones, eligiendo un tamaño de memoria grande para poder ir almacenando Opcodes. Debido a la configuración de las memorias, podemos hacer el Decode y el Read de las memorias en el mismo ciclo de reloj. Lo mismo ocurre con el Execute y el Write. En definitiva, cada instrucción hace en un ciclo de máquina el Fetch y en el siguiente se hace el Decode-Execute. Este método consigue un ciclo de máquina de dos ciclos de reloj y cada instrucción solo requiere un ciclo de máquina.

La opción seleccionada finalmente es esta última, trabajando en modo mixto con memoria de datos de 32 Bits. Se consigue un ciclo de máquina de dos ciclos de reloj y todas las instrucciones salvo excepciones tienen una duración de un ciclo.

3.2 Configuración de las memorias.

Otro factor muy importante que determina la velocidad del núcleo es el acceso a las memorias. La estrategia elegida en el apartado anterior requería poder leer y escribir en un solo ciclo, mientras se realizaba el Decode y el Execute.

Para abordar este problema solo hemos tenido en cuenta la configuración que debemos usar. Como las FPGA tienen bloques internos de RAM, no hay problemas de tiempos de acceso a las mismas.

Antes de continuar, es importante recordar que las memorias tienen un ancho de palabra que puede ser de 1 Byte, 2 Bytes, 4 Bytes... etc. La dirección del dato que el microprocesador pretende leer no será la misma que la dirección física en la memoria. Como el ancho de palabra de las memorias es exponente de 2, normalmente la dirección de memoria es la parte superior de la dirección global y la parte baja selecciona la posición de comienzo de lectura.

El problema que se nos presenta son las lecturas de varios Bytes que sobrepasan el ancho de las palabras de la memoria. Es decir, cuando hay que leer al menos dos líneas de memoria para obtener el dato completo. Este caso es inevitable debido a que los datos no están alineados y el comienzo de un dato puede estar en la parte alta de la dirección. A continuación se muestran unos ejemplos gráficos en una memoria genérica para poder ver con más claridad el problema:

DIRECCIÓN	BYTE 1	BYTE 2	.....	BYTE N-1	BYTE N
1	XX	XX	.....	XX	XX
2	XX	XX	.....	XX	XX

Como se observa, para la lectura de este dato hay que leer dos direcciones de Ram. Si estas dos direcciones están en el mismo bloque de memoria, habrá que hacer dos lecturas en dos ciclos.

Ejemplo 2: Lectura de 4 Bytes a partir de la posición N-1.

DIRECCIÓN	BYTE 1	BYTE 2	.....	BYTE N-1	BYTE N
1	XX	XX	.....	XX	XX
2	XX	XX	.....	XX	XX

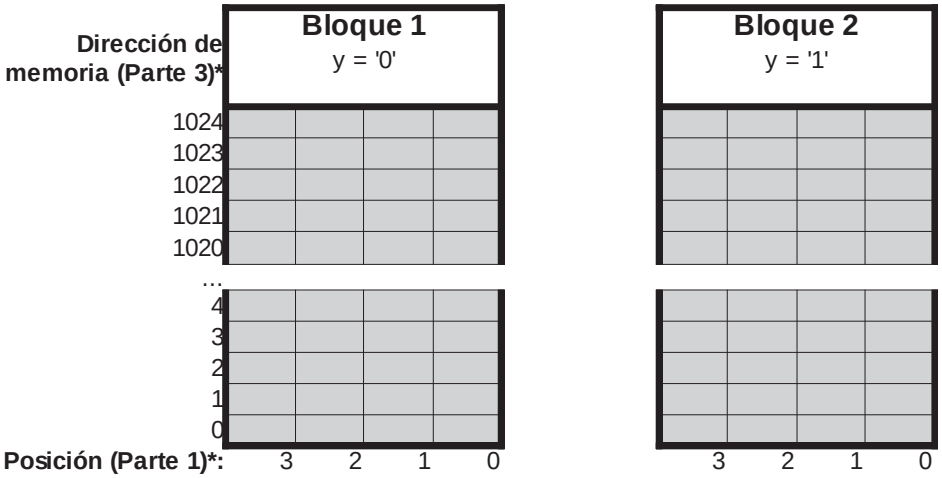
Observase que si la memoria es de 2 bytes se necesitaran hacer 3 lecturas.

DIRECCIÓN	BYTE 1	BYTE 2
1	XX	XX
2	XX	XX
3	XX	XX

La solución adoptada para estos problemas es poner dos bloques de memoria en paralelo. De esta forma podemos leer en dos direcciones consecutivas en un solo ciclo. Aun así queda pendiente el tamaño de palabra de las memorias. El tamaño mínimo para no tener problemas es igual al tamaño del dato más grande que se puede acceder, en este caso son 4 Bytes o 32 bits. Se pueden seleccionar memorias más grandes, pero no se consigue ningún beneficio adicional.

Nuestro espacio de memoria RAM está dividido en dos bloques de memoria de 4 Bytes de palabra. Para la lectura o escritura se coge la dirección dada por las instrucciones y se divide en tres partes: los dos primeros bits, el tercer bit y el resto de bits. Los dos primeros bits representan la posición del primer Byte en la palabra. El segundo selecciona el bloque de memoria y la tercera parte la fila. El tamaño de cada memoria es de 1024 direcciones, que nos ofrece un tamaño total de memoria RAM de 8KB. En la siguiente figura puede verse una representación del espacio de memoria.

Dirección: xxxxxxxxxxxxxxxxxxxxxx-y-zz
Parte 1: zz
Parte 2: y
Parte 3: xxxxxxxxxxxxxxxxxxxxxx



* Valores binarios puestos en base decimal.

Figura 3.2_1: Espacio de memoria RAM total de nuestro diseño y la estructura de las direcciones.

Fuente: Elaboración propia.

3.3 Bloques del núcleo.

El diseño del núcleo del procesador se ha realizado en 6 bloques interconectados entre sí. Cada proceso interno del microprocesador está representado por un bloque. Los dos restantes son bloques especiales que se ha decidido sacarlos fuera para facilitar el diseño.

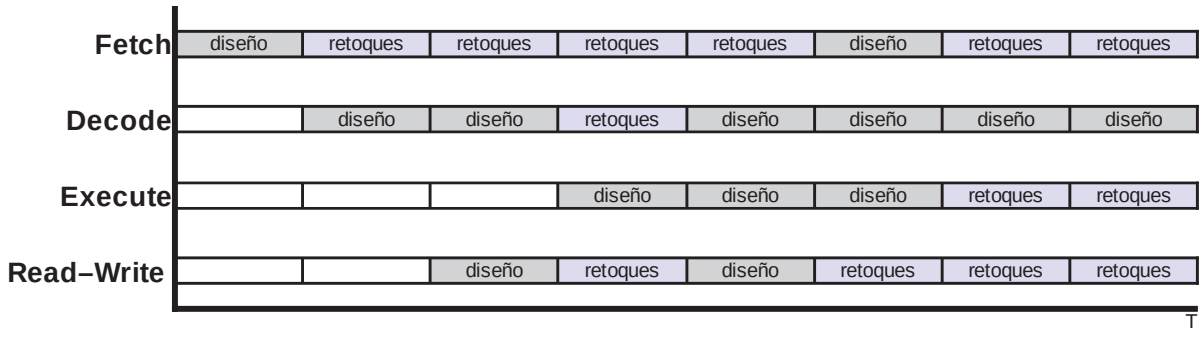


Figura 3.3_1: Diagrama de Gant del proceso de diseño.

Fuente: Elaboración propia.

Se ha seguido un orden de diseño de los bloques atendiendo a las necesidades que surgen en cada momento. Primero se ha diseñado el bloque del proceso Fetch, puesto que necesitamos los Opcodes de las instrucciones. Después, se ha diseñado el bloque del proceso Decode. Tras este proceso se ha diseñado el bloque de control de las memorias RAM y de los registros. Este bloque se encarga tanto de leer como de escribir las memorias. Cuando ya se han tenido disponibles los datos, se ha procedido con el diseño del bloque encargado de realizar el Execute.

Todos los bloques se han diseñado por partes, alcanzando unos objetivos que se han propuesto dependiendo de las necesidades puntuales del conjunto. Conforme se ha ido avanzando con el diseño se han requerido nuevas funciones en algunos bloques. Como se ve en la figura 3.3_1, todos los bloques han tenido varias fases de diseño y siempre se han ido haciendo pequeños cambios para ajustar ciertas señales. Cuando se indica retoques en la figura, se refiere a esos pequeños cambios. Con diseño se refiere a la introducción de nuevas funciones que antes no estaban y que el estado del diseño solicita.

Los nombres de cada bloque y su correspondiente función son:

- **Fetch_c2:** Se encarga de realizar el Fetch.
- **Decode_c2:** Realiza el Decode.
- **Alu:** Realizar el proceso Execute.
- **Reg_block_c2:** Lee y escribe los registros y la RAM.
- **Diviblock2:** Realiza la operación de la división. No se realiza en el bloque Alu por motivos que más adelante se explicaran.
- **Registros_PSW:** Contiene los registros PSW. No están con el resto de los registros SFR debido al gran número de ocasiones que se modifica.
- **Ramsp1kx32:** Bloques de la memoria RAM. Las FPGA´s tienen en su interior bloques de RAM configurables. Este bloque simula el comportamiento de la memoria que hemos elegido. No son sintetizables, solo sirve para realizar simulaciones y ver cómo se comporta la memoria.

En la figura 3.3_2 podemos ver los bloques que forman el núcleo. Se corresponde al tercer nivel de los esquemáticos.

La descripción de los bloques, Packages y otras entidades se encuentra en el anexo 3. Aquí se ven los procesos en los que están dividido cada bloque. Se explica la función de cada uno. También muestra los algoritmos más complicados haciendo una breve descripción de ellos.

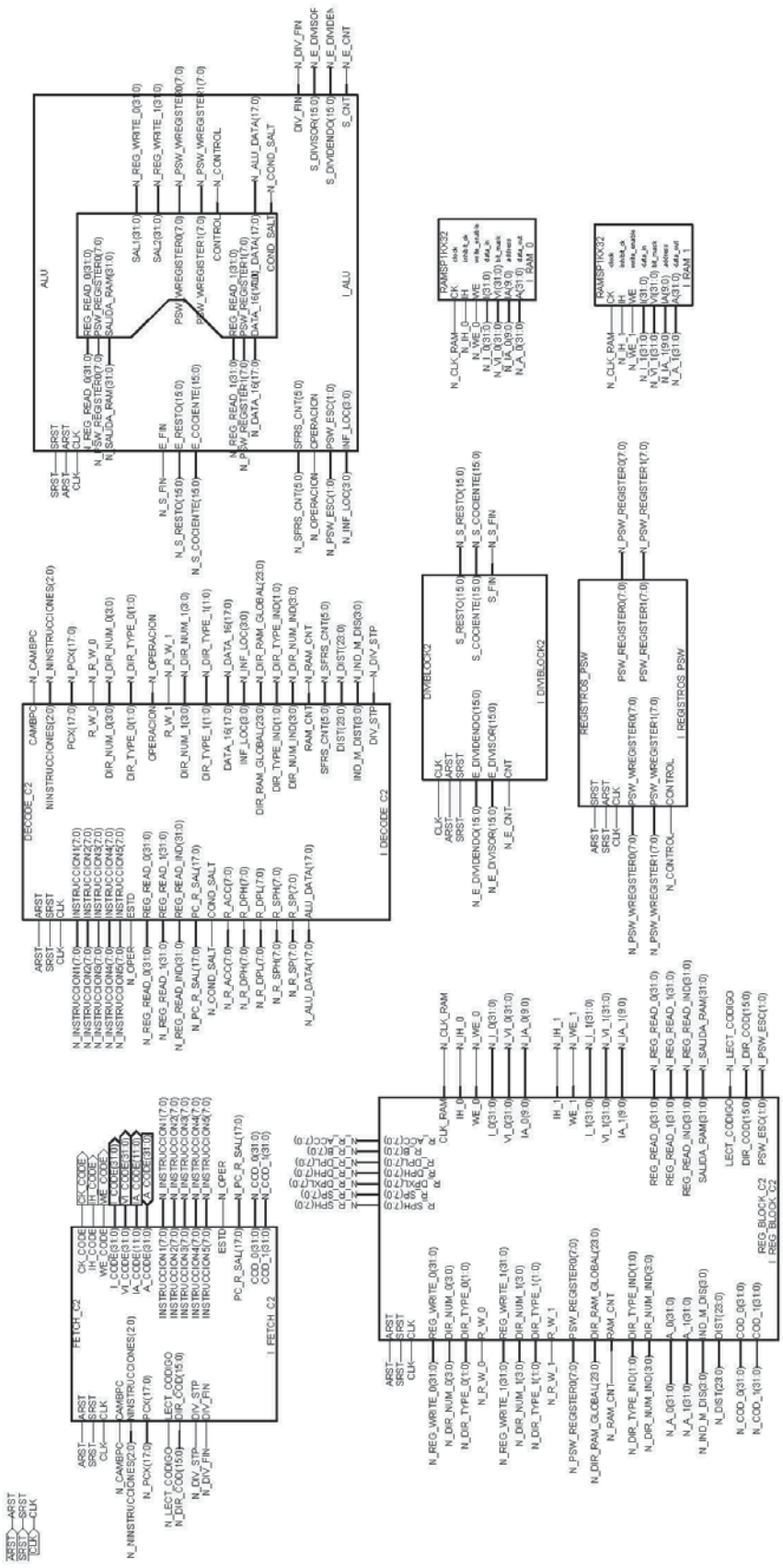


Figura 3.3_2: Esquemáticos de todos los bloques presentes en el núcleo.

Fuente: Elaboración propia.

Capítulo 3: Diseño del núcleo del microprocesador 80251.

Capítulo 4: Pruebas y resultados.

En esta sección vamos a ver las operaciones que se han realizado para comprobar el diseño. Se empiezan con la verificación del código para comprobar que funciona correctamente. Posteriormente se comparará el resultado de una simulación en un compilador del 251 con el resultado de nuestro microprocesador. Finalmente se hace una prueba de síntesis para comparar los recursos usados con los que usa el 8051 sin periféricos.

4.1 Comprobación del código VHDL.

Las pruebas se han realizado mediante simulaciones en Modelsim 6.2. Cada bloque ha sido probado independientemente. Como se ha comentado en el capítulo anterior, la mayoría de bloques han sido diseñados en un momento determinado y luego completados o ajustados. Implica que las pruebas de los bloques se hayan tenido que ir haciendo periódicamente.

Se han creado varios proyectos paralelos en donde probar las primeras versiones de los bloques. En ellos se ha introducido el bloque de prueba en un estado de diseño bastante avanzado. También se han creado bloques de Test que generan señales que son enviadas a los bloques de prueba. A las señales generadas en estos bloques se les denomina Testbench y se busca que abarquen todo el rango de utilidad del diseño de prueba. Al probar todas las combinaciones posibles se verifica que funcionara correctamente al colocar el bloque en el proyecto definitivo.

Algunas pruebas requieren además del bloque de test, otros bloques ya diseñados. No hay problemas en introducir en la pruebas cuantos bloques se necesiten, pero tenemos que tener cuidado a la hora de hacer modificaciones en estos. Si introducimos alguna modificación se deben pasar las pruebas de nuevo.

Aunque las primeras pruebas se hacen de forma individual en proyectos paralelos, las pruebas finales se deben hacer en el proyecto principal. Incluso algunos bloques se prueban directamente desde el proyecto principal. Tal es el caso del bloque Alu que no tiene sentido probarlo individualmente cuando necesita todos los bloques.

El bloque Decode, que es el más grande y el que ha requerido más modificaciones, ha sido probado de forma mixta. Se ha creado un sub-proyecto en el que se han probado las primeras fases del diseño. Pero las últimas fases, entre las que están las asignaciones de todas las instrucciones, han sido probadas en el proyecto principal. Las pruebas en este caso se han realizado al mismo tiempo en que se diseñaban. Los Testbench en este punto no son señales, sino que se han creado vectores de Opcodes que lee el Fetch para probar directamente las instrucciones. Se mira que el Decode genera bien las señales de control y que el Execute las interpreta correctamente. Se han probado las 269 instrucciones en los dos modos de funcionamiento. Para cada instrucción se han realizado las pruebas necesarias para abarcar todas las posibilidades que tienen. Si la instrucción modifica los Flags se han hecho pruebas para que los modifique todos y comprobar que no cambia algún Flag no deseado.

Para realizar las pruebas se deben compilar los archivos de los bloques con Modelsim. También deben ser compilados los Packages y otras entidades. Para realizar la compilación se ha creado un programa que permite compilar todos los archivos a la vez y no tener que ir uno por uno con cada archivo. Una vez compilado debemos seleccionar las señales que pretendemos comprobar e introducirlas en el visor de señales. Por último se escribe el tiempo de simulación y se comprueba el resultado en forma de señales.

Por lo general los resultados de las simulaciones son como se pueden ver en la figura 4.1_1. La simulación aporta información que nos permite verificar que los datos se copian correctamente en el destino, podemos ver si las operaciones aritméticas dan el resultado correcto y cambian bien los Flags, ver si las comparaciones entre variables se hacen correctamente en el momento adecuado....

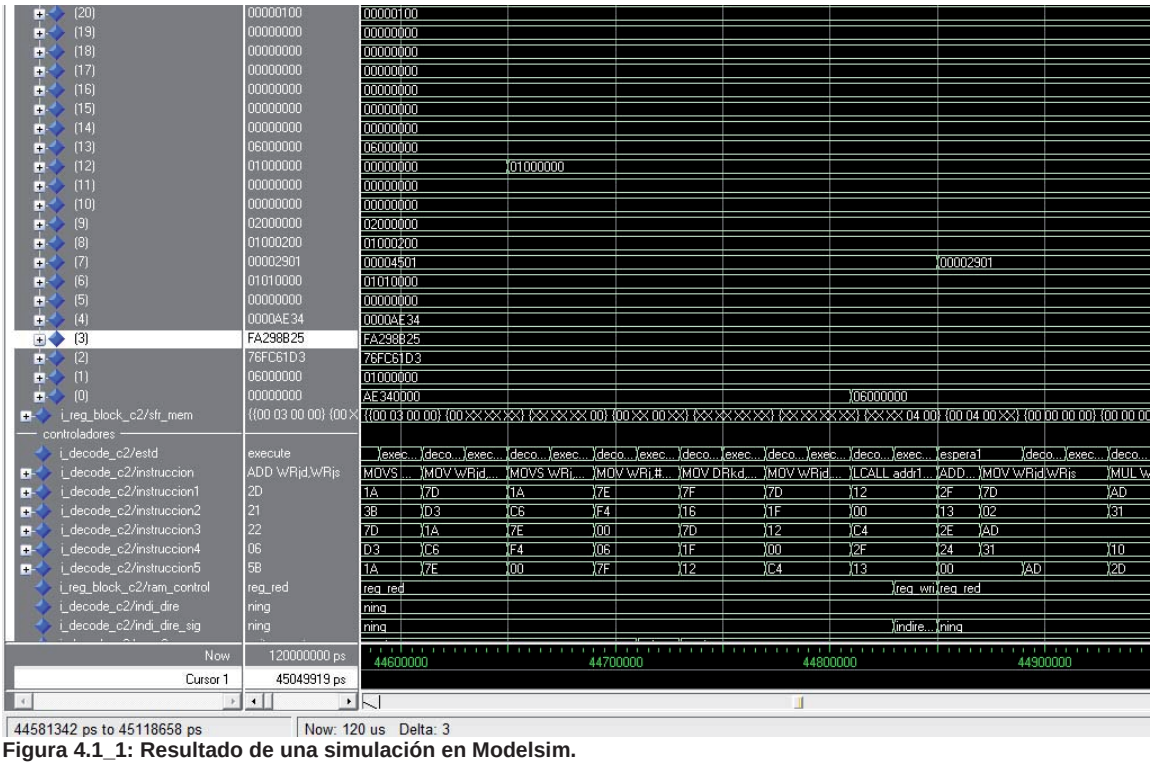


Figura 4.1_1: Resultado de una simulación en Modelsim.
Fuente: Elaboración propia.

4.2 Verificación del código VHDL.

Una cosa es que nuestro diseño funcione bien como lo hemos hecho y otra es que el funcionamiento se ajuste al diseño original. El principal motivo por el cual nuestro diseño no funciona igual al original es que el Datasheet no es muy explicito a la hora de describir algunas instrucciones y además contiene erratas. Para comprobar que funciona igual que el 251 original se debe comparar el resultado que ofrece nuestro diseño con el resultado que daría el 251. La comparación se hace ejecutando el mismo programa en un compilador y en nuestro diseño. Hemos usado el compilador Keil uvision4 para la verificación del código. Se escribe un programa en C en el compilador y este se encarga de compilarlo y generar un archivo en hexadecimal que cargaremos en nuestro diseño.

Se han programado en C una serie de programas sencillos que ejecutan operaciones matemáticas. Entre ellos, uno que multiplica dos matrices 3x3 como se muestra a continuación y que va a ser el ejemplo desarrollado en esta sección:

```c
#include <math.h>

const int MATRIZ1[3][3] = {1627, 3248, 5642, 6743, 999, 1, 23, 4563, 11}; //00-01-10-11
const int MATRIZ2[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9}; //00-01-10-11
int MATRIZ3[3][3] = {0,0,0,0,0,0,0,0,0};
char I, J, K;

main()
{

 for (I=0; I<3; I++)
 {
  for (J=0; J<3; J++)
  {
   for (K=0; K<3; K++)
   {
    MATRIZ3[I][J] += MATRIZ1[I][K] * MATRIZ2[K][J];
   }
  }
 }


}
```

Figura 4.2_1: Código en C para la multiplicación de dos matrices.
Fuente: Elaboración propia.

$$\begin{pmatrix} 1627 & 3248 & 5642 \\ 6743 & 999 & 1 \\ 23 & 4563 & 11 \end{pmatrix} * \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 54113 & 64630 & 75147 \\ 10746 & 18489 & 26232 \\ 18352 & 22949 & 27546 \end{pmatrix}$$

Una vez simulado, Keil nos permite ver los datos guardados en la memoria RAM, registros y SFR´s. Para comprobar nuestro diseño debemos cargar el programa en hexadecimal y mirar si al final de la simulación ambos resultados son iguales.

En las figuras 4.2_3 y 4.2_4 podemos ver dos pantallazos, el primero del Keil y el segundo del Modelsim. Se corresponden al final de la ejecución del programa. En el Keil se ve la memoria RAM en un lado y en el otro los registros. En la figura 4.2_4 se ve representado en Modelsim el banco de registros.

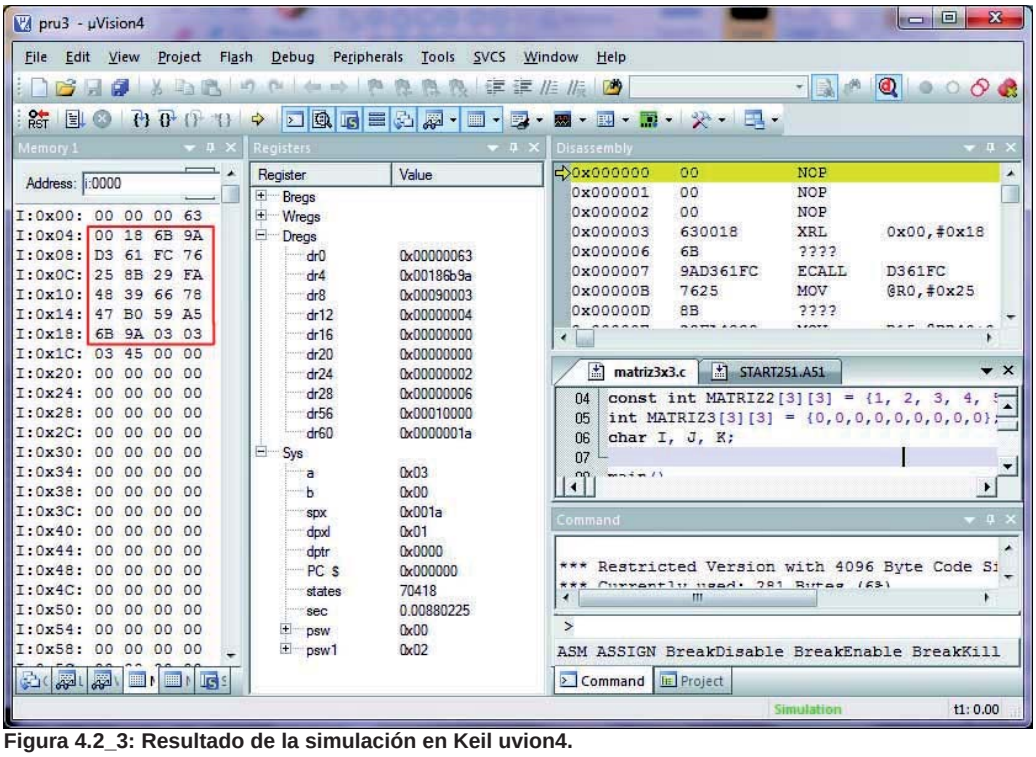


Figura 4.2_3: Resultado de la simulación en Keil uvion4.
Fuente: Elaboración propia.

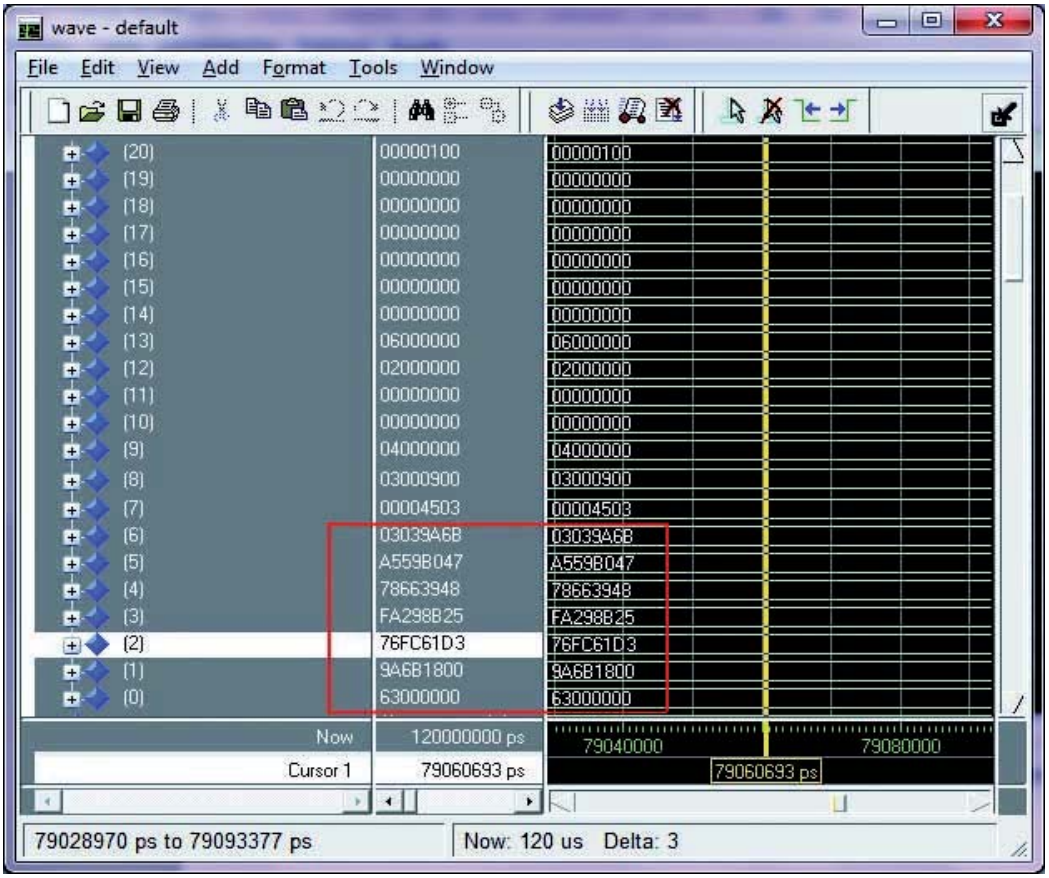


Figura 4.2_4: Visor de ondas del Modelsim en la última instrucción del programa.
Fuente: Elaboración propia.

La solución de la matriz es el recuadro rojo de las imágenes. En Keil el número de la izquierda es el de menor peso mientras que el de la derecha es el de mayor peso. Las direcciones van desde arriba hacia abajo. La simulación en Modelsim va al revés, de derecha a izquierda y de abajo a arriba. Para que quede más claro, el valor 63 de la primera línea del Keil y de la última de Modelsim se corresponde al cuarto Byte, R4.

Como ambos resultados son iguales y el resto de parámetros también, se puede asegurar que ambos se han comportado de la misma manera.

4.3 Pruebas de síntesis.

Las pruebas de síntesis se han realizado durante la parte final del diseño para comprobar que no haya código no sintetizable, el tamaño del diseño, y el ciclo de reloj. Hacer pruebas periódicas viene bien para depurar pequeños fallos, completar las listas de sensibilidad o eliminar señales que se han declarado pero que al final no ha sido necesario usarlas.

Una vez finalizado, además de informarnos de la frecuencia máxima de reloj que podemos usar y los recursos que consume de la FPGA, se encarga de convertir el código VHDL en hardware y asignar a los recursos de la FPGA la configuración necesaria.

Hemos usado para esta tarea el programa Synplify Pro 9.2 que además de los antes mencionado es capaz de reducir el tamaño del hardware.

En la siguiente imagen se muestra la FPGA que hemos usado para las pruebas de síntesis.

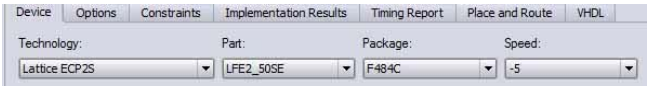


Figura 4.3_1: FPGA usada en las pruebas de síntesis.
Fuente: Elaboración propia.

Los resultados de las pruebas son:

Resource Usage Report Part: lfe2_50se-5	
Register bits:	2124 of 48000 (4%)
I/O cells:	268
DSP primitives:	1
Details:	
CCU2B:	393
FD1P3AX:	1437
FD1P3AY:	60
FD1P3DX:	2
FD1S3AX:	509
FD1S3AY:	108
FD1S3DX:	8
GSR:	1
IB:	35
INV:	4
L6MUX21:	320
MULT18X18B:	1
OB:	233
ORCALUT4:	12925
PFUMX:	1383
ROM256X1:	22
VHI:	1
VLO:	1

Tabla 13: Recursos de la FPGA usados con arquitectura del 251.
Fuente: Elaboración propia.

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
U80251C2_CORE|CLK	38.8 MHz	25.4 MHz	25.748	39.371	-13.623	inferred	Autoconstr_clkgroup_0

Tabla 14: Informe de tiempos. Estima la frecuencia máxima del reloj.
Fuente: Elaboración propia.

4.4 Comparativa entre arquitecturas.

Para comprobar los beneficios logrados trabajando con la arquitectura del 251 respecto a la del 51, se ha comparado los resultados de la simulación del programa de verificación antes mostrado (figura 4.2_1) funcionando en ambas arquitecturas. Para ello se ha compilado el programa en un compilador de esa arquitectura y se ha simulado en el diseño anterior del microcontrolador. Se han usado tres versiones del programa de prueba, declarando las variables como carácter (char), entero (int) o entero largo (long int). El primero trabaja con datos tipo Byte y seria el nativo de la arquitectura del 51. Los otros dos trabajan con Words y con Double-word. De la simulación se observa los ciclos de reloj que tarda y el tamaño del código resultante.

Arquitectura	8051			80251				
Tipo de variable	Tclk	Data	Code	Tclk	Data	Code	Reducción de código.	Reducción del tiempo de ejecución
Char	13992	30	411	3182	33	266	32,20%	77,26%
Integer	20445	57	515	3344	60	273	41,78%	83,64%
Long Integer	36915	111	716	4736	114	329	46,43%	87,17%

Tabla 15: Resultados obtenidos tras ejecutar un programa con variables en formato “Char”, “Int” y “Long Int”. Fuente: Elaboración propia.

Se comprueba que efectivamente el tamaño del código resultante es aproximadamente la mitad. Si miramos el tamaño resultante de las tres pruebas se observa que trabajar con datos más grandes y con más instrucciones de direccionamiento disminuye el tamaño de código cerca del 40%. Respecto al tiempo se produce una notable mejoría. Ejecuta menos instrucciones con menos ciclos de reloj por lo que el tiempo empleado es entre 5 a 6 veces menor.

En la tabla 16 nos encontramos con los recursos usados en el núcleo del 8051 sin periféricos:

Resource Usage Report	
Part: lfe2_50se-5	
Register bits:	1169 of 48000(2%)
I/O cells:	287
Details:	
CCU2B:	199
FD1P3AX:	945
FD1P3AY:	123
FD1S3AX:	40
FD1S3AY:	5
GSR:	1
IB:	96
INV:	1
L6MUX21:	50
OB:	191
OFS1P3BX:	54
OFS1P3DX:	2
ORCALUT4:	4486
PFUMX:	315
ROM256X1:	71
VHI:	1
VLO:	1

Tabla 16: Recursos usados por el núcleo del microprocesador 8051.

Fuente: Elaboración propia.

Si lo comparamos con los recursos usados por nuestro diseño, este resulta mayor. Era lo esperado debido a que la arquitectura del 251 tiene más registros, más del doble de instrucciones y estas pueden operar con datos mayores. Además debemos tener en cuenta que la reducción del tiempo de ejecución de las instrucciones ha sido a costa de aumentar el hardware.

Para finalizar, el diseño se ha procesado con Synopsys para conocer de forma aproximada cuanto espacio ocuparía en silicio. Este programa hace síntesis del diseño en VHDL de forma más restrictiva que Synplify, pero lo que nos interesa es que permite implementar nuestro diseño como un ASIC. El núcleo del microcontrolador sin memorias se descompone por un lado en la parte de biestables y por el otro en la lógica combinacional. El programa calcula la velocidad máxima a la que podrá funcionar el diseño, que será siempre mayor que en una FPGA implementada con la misma tecnología.

El resultado de Synopsys nos viene en UA´s, unidades de área, que dependiendo de la tecnología que utilicemos se convierte en un área en mm^2. A partir de estas UA´s se calculan las puertas equivalentes de nuestro diseño. Para hacer una puerta NAN2 se necesitan 35 UA´s. En nuestro caso trabajamos con una tecnología de 90nm y las UA´s son 0,28x0,28 um. Synopsys detecta el camino crítico. Se le fija un determinado tiempo para realizar dicho camino y realiza una serie de iteraciones para conseguir aproximarse a él. Tras finalizar estas iteraciones el programa nos muestra el tamaño del Hardware y el tiempo mínimo necesario para realizar el camino crítico. A partir de este tiempo se calcula la frecuencia máxima que podremos usar en el diseño.

El resultado obtenido es el mostrado en la tabla 17 en unidades de área:

Tamaño total:	1620430 UA
Combinacional:	1202908 UA
Registros:	417523 UA
Espacio total usado:	0,127 mm^2
Tiempo de periodo:	16 ns
Frecuencia máxima:	62,5 MHz
Puertas equivalentes:	46298 Peq

Tabla 17: Resultados en unidades de área y nanosegundos obtenidos al procesar el diseño con Synopsys. A partir de este dato se han calculado el espacio total en el circuito, la frecuencia máxima del reloj y el número de puertas equivalentes. Fuente: Elaboración propia.

El tamaño de las memorias internas, tanto RAM como Flash, dependen del fabricante del integrado. El fabricante suministra un catalogo con las opciones disponibles y que nos permiten cierta libertad de diseño. Dependiendo del tamaño de las memorias y con la ayuda de unas tablas se calcula su tamaño.

En nuestro caso, tenemos dos memorias de 1024X4 Bytes. Buscando en un catalogo encontramos el tamaño que ocupara en el ASIC. En la tabla 18 vemos el espacio ocupado por las dos memorias:

Espacio en UA	
Espacio de una memoria Ram de 1kX32:	**980565 UA**
Espacio de las dos memorias:	**1961130 UA**
Espacio en mm^2	
Espacio de las dos memorias:	**0,1537 mm^2**

Tabla 18: Espacio de las memorias RAM.
Fuente: Elaboración propia.

Capítulo 5: Conclusiones.

5.1 Conclusiones sobre el Proyecto.

En este proyecto hemos estudiado el funcionamiento de los microprocesadores que encontramos en los núcleos de los microcontroladores de la familia 80251. De ellos hemos analizado sus principales características funcionales. Hemos visto que el espacio de memoria se divide en cuatro partes, una reservada para código de los programas, otra es memoria RAM interna y las otras nos permiten usar memorias externas, tanto para programas como para RAM. El acceso a este espacio de memoria se puede hacer con tres tipos de direccionamiento, directo, indirecto y direccionamiento de bits. Cada uno permite un rango de acceso y un tamaño de datos. Los datos pueden ser de 1 Byte, 2 Bytes y 4 Bytes, o lo que es lo mismo, Bytes, Word, Double-words. Los datos en formato Word y Double-word son almacenados en memoria en formato big-endian que posiciona el Byte de mayor peso en la posición más baja y el Byte de menos peso en la posición más alta.

Cuenta con dos bancos de registros, uno dedicado a funciones de control de periféricos y variables especiales denominados SFR´s, y otro banco con registros para almacenar datos, que comparte la parte inferior de los mismos con la parte baja de la memoria RAM interna. Por cuestión de compatibilidad con sus predecesores, los microcontroladores 8051, tiene que trabajar en dos modos de funcionamiento, modo binario compatible con la arquitectura del 51 y el modo fuente. Según el modo de funcionamiento el mapa de instrucciones cambia para que los programas de la anterior tecnología puedan trabajar sin tener que reprogramarlos.

Conociendo el funcionamiento básico de la arquitectura del 251 se ha procedido al diseño del microprocesador. Se ha empezado mirando las fases por las que pasa la

ejecución de las instrucciones. Finalmente se ha optado por realizar el Fetch de forma independiente, el Decode-Read y el Execute-Write se realizan secuencialmente.

La configuración de las memorias internas para lograr el menor ciclo ha sido poner dos memorias en paralelo de cuatro Bytes de ancho de palabra para las memorias RAM, mientras que la memoria del código de programa es también de cuatro Bytes de palabra pero tan solo una memoria. El diseño se ha realizado a partir de unos esquemáticos, que por una parte representan el proyecto desde la placa hasta el núcleo, y por otra representa dentro del núcleo los diferentes bloques que han sido diseñados. El núcleo se ha dividido en 6 bloques, 4 de ellos se encargan de los procesos internos del microprocesador y los otros dos los complementan con funciones especiales.

Los bloques han sido compilados y puestos a prueba con el programa Modelsim. Con el compilador Keil uvision 4 se ha comparado la forma de funcionamiento de nuestro microprocesador con uno comercial, mostrando ambos el mismo resultado tras realizar las simulaciones. Por último, se han hecho pruebas de síntesis que han pasado satisfactoriamente.

Este diseño tiene las siguientes características:

- Dos ciclos de reloj por ciclo de máquina.
- Frecuencia de reloj máxima de 25 MHz, frecuencia de máquina de 12.5 Mips.
- Reducido número de instrucciones que requieren más de un ciclo de máquina para ejecutarse.
- La memoria RAM interna está dividida en dos bloques de memoria en paralelo con tamaño de palabra de 4 Bytes.
- Un bloque de memoria ROM para código con tamaño de palabra de 4 Bytes.
- El tamaño del Hardware generado nos permite introducirlo en la FPGA sobrando suficientes recursos para periféricos.

5.2 Futuras líneas de desarrollo.

Del microcontrolador solo se ha diseñado el núcleo, sin periféricos ni interrupciones. Para poder contar con una IP completa del microcontrolador 80251 se deberían diseñar los periféricos e introducirlos en el diseño.

El diseño de la mayoría de los periféricos se puede hacer de forma independiente de nuestro diseño. El control de estos dispositivos se hace mediante los registros SFR, así como la lectura o transmisión de datos. El diseño se debería hacer en un bloque independiente junto al bloque del núcleo (segundo nivel de los esquemáticos), y solo habría que sacar del bloque Reg_block_c2 señales de control hasta este nivel.

Para implementar los puertos habría que hacer más modificaciones dentro de nuestro diseño. Los puertos son bidireccionales, con lo que pueden funcionar como

entradas o salidas, y van desde el núcleo hasta la placa. Se tendrá que tener en cuenta que algunos periféricos usan pines de los puertos como entradas o salidas.

Por último está el tema de las interrupciones. Los periféricos en muchas ocasiones informan al microprocesador que han terminado o se han producido cambios mediante interrupciones. De esta forma evitamos tener que leerlos periódicamente y se gana en eficiencia. Además se evita la perdida de información por tardar demasiado en leer un dato. Las interrupciones detienen la ejecución del programa realizando un salto a una determinada dirección. Las interrupciones ejecutan una rutina y cuando la terminan continúan con la ejecución del programa donde se había quedado. Para ello antes de realizar el salto guardan el Program Counter (PC). La gestión de interrupciones se hace mediante SFR´s permitiendo habilitar las interrupciones y darles prioridad.

Otra vía de desarrollo puede ser mejorar el ciclo de máquina que dure un ciclo de reloj, siguiendo un modo de trabajo Pipeline. Este modo de trabajo necesitará implementar una memoria de código similar a nuestra memoria RAM, que permita leer las instrucciones completas en un ciclo de reloj. También se deberá diseñar el Hardware pensado para poder leer las instrucciones antes de ser escritas en memoria, debido a que aparecerán dependencias entre instrucciones. Es ente caso la verificación será mucho más complicada, al tener las instrucciones dependencias entre ellas.

5.3 Conclusiones personales.

La realización de este proyecto me ha supuesto un gran reto a la hora de diseñar con VHDL. Comprender el funcionamiento del microcontrolador para desarrollar una IP sintetizable no ha sido una tarea fácil. En algunos casos encontrar la solución a un problema me ha supuesto un gran esfuerzo pero al final siempre he conseguido encontrar una solución adecuada. A veces, las partes que parecen más sencillas y obvias son las que más dificultad me han supuesto.

Este proyecto no sólo ha enriquecimiento mi conocimiento sobre el diseño con VHDL, también he aprendido a usar otros programas. Gracias a este proyecto he aprendido a dividir diseños en bloques. También he mejorado el uso de Modelsim, que lo manejo con mayor soltura, y la realización de esquemáticos. Los programas Keil y Symplify 9.2 nunca los había usado y he tenido que aprender a usarlos.

Estoy muy satisfecho con el trabajo realizado. Después de tanto tiempo y esfuerzo ver funcionar el diseño es muy gratificante. Aunque con la experiencia adquirida haría las cosas de diferente manera. Lo cual creo es que algo normal, ya que ha sido la primera vez que me he enfrentado con un microcontrolador por dentro, y conforme el proyecto ha ido avanzando me he encontrado con problemas que he tenido que resolver.

Capítulo 5: Conclusiones.